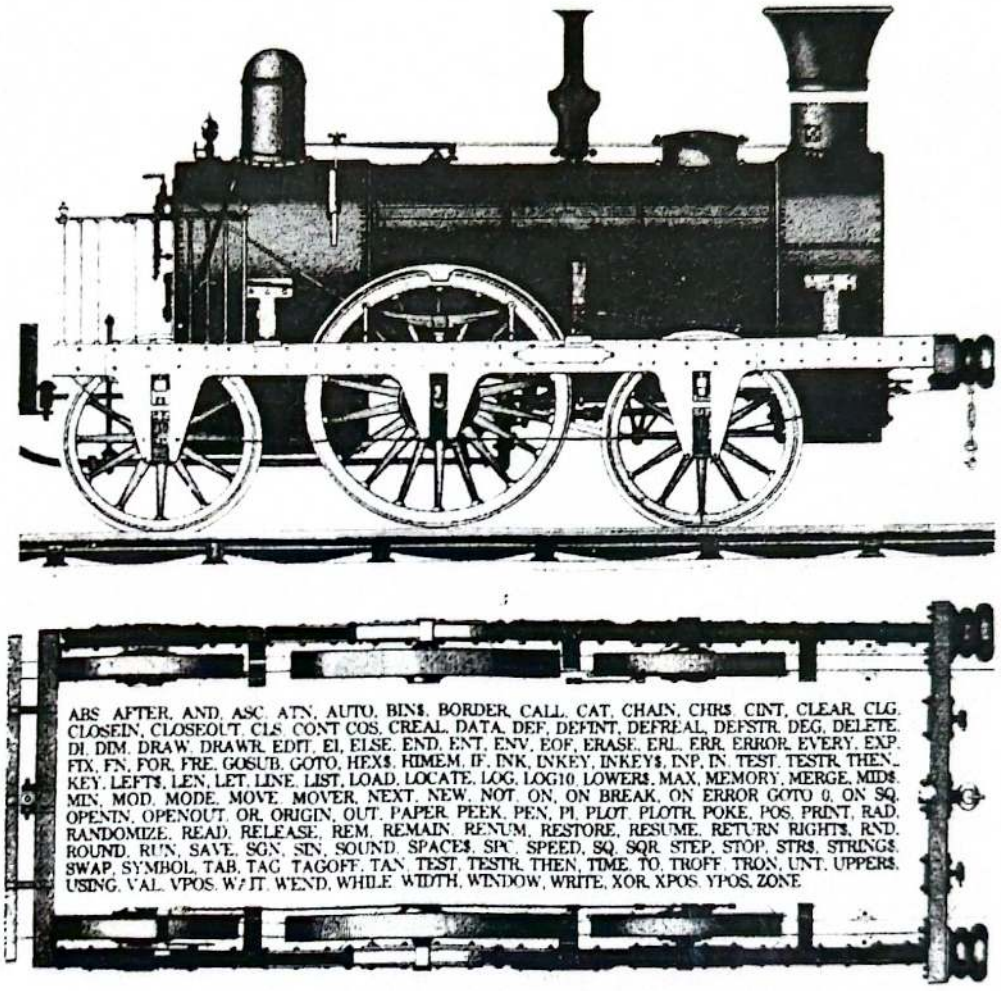
Amstrad BASIC:

The Complete Technical Specification


ABS, AFTER, AND, ASC, ATN, AUTO, BIN$, BORDER, CALL, CAT, CHAIN, CHR$, CINT, CLEAR, CLG, CLOSEIN, CLOSEOUT, CLS, CONT, COS, CREAL, DATA, DEF, DEFINT, DEFREAL, DEFSTR, DEG, DELETE, DI, DIM, DRAW, DRAWR, EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE, ERL, ERR, ERROR, EVERY, EXP, FIX, FN, FOR, FRE, GOSUB, GOTO, HEX$, HIMEM, IF, INK, INKEY, INKEY$, INP, IN, TEST, TESTR, THEN, KEY, LEFT$, LEN, LET, LINE, LIST, LOAD, LOCATE, LOG, LOG10, LOWER$, MAX, MEMORY, MERGE, MID$, MIN, MOD, MODE, MOVE, MOVER, NEXT, NEW, NOT, ON, ON BREAK, ON ERROR GOTO 0, ON SQ, OPENIN, OPENOUT, OR, ORIGIN, OUT, PAPER, PEEK, PEN, PI, PLOT, PLOTR, POKE, POS, PRINT, RAD, RANDOMIZE, READ, RELEASE, REM, REMAIN, RENUM, RESTORE, RESUME, RETURN, RIGHT$, RND, ROUND, RUN, SAVE, SGN, SIN, SOUND, SPACE$, SPC, SPEED, SQ, SQR, STEP, STOP, STR$, STRING$, SWAP, SYMBOL, TAB, TAG, TAGOFF, TAN, TEST, TESTR, THEN, TIME, TO, TROFF, TRON, UNT, UPPER$, USING, VAL, VPOS, WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE, XOR, XPOS, YPOS, ZONE

Published by AMSOFT, a division of
Amstrad Consumer Electronics plc
Brentwood House
169 Kings Road
Brentwood
Essex

All rights reserved
First edition 1984
Reproduction or translation of any part of this publication without permission of the copyright owner is unlawful. Amstrad and Locomotive Software reserve the right to amend or alter the specification without notice. While every effort has been made to verify that this complex software works as described, it is not possible to test any program of this complexity under all possible conditions. Therefore Locomotive BASIC is provided "as is" without warranty of any kind either express or implied.

SOFT157 ISBN 1 85084 001 6
Copyright © 1984 Locomotive Software and Amstrad Consumer Electronics plc

Foreword

This complete and concise description of Locomotive Software's Mallard BASIC interpreter as implemented for the AMSTRAD CPC464 is intended to supplement the introduction in the CPC464 user's guide. It is not guide to BASIC programming, but an important reference work that establishes the ground rules and usage of the language.

BASIC features have been described carefully using a formal terminology, and you should read the description of this metalanguage before proceeding to the main body of the description.

Pages are identified using the form:

(Section).(Page Number)

use of CAPITAL letters, since many of the extension features are described elsewhere in this book in the sections at the front or rear, in which context the index identifies them using upper/lower case.

Remember that each keyword description contains a list of associated keywords, further adding to the cross reference of the contents.

Please also note that a companion publication is available (Autumn 1984) describing the CPC464 operating system and firmware (SOFT 158), and we strongly advise machine code and advanced BASIC devotees to add this to their reference library.

Notes on style

Please note that the typestyles used in AMSOFT publications are intended to help identify the different operations and sequences used in computer operation:

Text that is typed at the keyboard and appears on the screen is shown in an OCR-B typeface:

```
10 FOR N= 1 TO 50
```

Keyboard actions that instruct a command sequence but do not necessarily have a corresponding representation on the screen are shown in Helvetica 75 typeface. Non-printing keys are shown enclosed in square brackets:

P	Where the letter 'P' appears on the screen display
[P] **[ESC]**	As a command, with no corresponding displayed character

General narrative and descriptive text will be shown in one of a variety of serif typestyles, eg: Century, Palatino, Times etc.

CONTENTS

Introduction 1.1
Elements of Basic 2.1
Direct and Program Mode 3.1
The Screen 4.1
Keyboard and Joystick 5.1
Line Input, Edit and Copy 6.1
Streams 7.1
Sound Generation 8.1
BASIC Interrupts 9.1
Overview of Commands 10.1
Commands and Built In Functions 11.1
BASIC Keywords in alphabetical order A.1
Error Numbers and Error Messages APP1.1
External Routines and Commands APP2.1
BASIC Keywords (summary) APP3.1
Text Screen Legal Position & Control Characters APP4.1
Keyboard and Joystick Key Numbers APP5.1
Colours APP6.1
Notes and Tone Periods APP7.1
Cassette Reading and Writing APP8.1
Memory,Himem,Symbol After and Cassette Buffers APP9.1
INDEX APP10

1. Introduction.

The BASIC interpreter is capable of executing the commands given in Section 11. Each command is identified by one or more leading keywords, and may have a number of parameters. In general each parameter may be an expression, involving constants, variables and functions. Strings and various forms of numeric data types are supported, as is sequential file handling.

Commands are presented to BASIC on lines. A line may comprise several commands, separated by colons, limited only by the line length. In Direct Mode lines are input from the keyboard. In Program Mode lines are read from the current program in memory.

In Direct Mode it is possible to add and remove lines from the program, and to amend existing lines.

1.1 Line Input.

BASIC accepts lines from the console of up to 255 characters, terminated by a carriage return. During line input it is possible to edit the current line, and to use the 'Copy' cursor facilities to insert characters from elsewhere on the screen.

1.2 Metalanguage.

In order to describe commands and their parameters a simple metalanguage is used. The form of each command is described as it appears when entered, with any variable or optional parts shown by 'place-holders' which refer to a concept defined elsewhere.

A concept is represented by its name enclosed in angle brackets. For example, in various places an expression yielding a numeric value is required, this is represented by :

```
<numeric expression>
```

Anything not enclosed by angle brackets is required as given. For example the **STOP** command takes the form :

```
STOP
```

Where there is an optional part in a definition, the optional part is enclosed in square brackets. For example, if a numeric expression were to be optional then it would appear :

```
[<numeric expression>]
```

If an optional part may be repeated (so it may appear any number of times, including none at all) an asterisk is appended after the closing square bracket. For example, a string of digits, requiring at least one digit, would appear :

‹digit›[‹digit›]*

In many places a list of objects separated by commas is used. A short form is used, which is best illustrated by example, thus :

‹list of: ‹expression› is : ‹expression›[, ‹expression›]*
or: ‹list of: [#]‹number› is: [#]‹number›[, [#]‹number›]*

Note that the list may be a single object. If the list contains more than one object, then each additional object must be preceded by a comma.

2. The Elements of BASIC.

2.1 Character Set.

Characters in the machine have values in the range 0..255 (so a character is stored in a byte). Characters in strings may take any value. In general, however, BASIC assumes that the ASCII character set is in use..

Characters with values less than 32 (&20, space) are treated as non-printing, and several have special meaning to BASIC. Characters with values greater than 126 (&7E, *tilde*) are generally ignored.

When processing command and program lines BASIC does not distinguish between lower case and upper case, except in strings.

2.2 White Space.

Spaces and all non-printing characters other than carriage return (or any combination thereof) count as ‹white space›. ‹White space› is not significant in command lines, except where it has the effect of terminating an item.

2.3 Names.

Variables and Keywords have ‹name›s. The first character of a ‹name› must be alphabetic, further characters may be alphabetic, numeric or dot. Names may be at most forty characters long, all of which are significant. A ‹name› is terminated by any character which may not be part of a ‹name›. BASIC does not distinguish between upper and lower case characters in ‹name›s.

Keywords generally take the form of ‹name›s, but may include other characters. Some keywords are expressed as more than one ‹name› each separated by white space.

Variables may not have the same name as a keyword.

2.4 Numbers.

A ‹number› may be any of the following :

‹unscaled number›
‹scaled number›
‹based number›
‹line number›

An ‹unscaled number› may take any of the following forms :

‹digits›
‹digits› . [‹digits›]
[‹digits›] . ‹digits›

where ‹digits› is one or more decimal digits. ‹White space› is allowed in ‹unscaled number›s, and may be used at will as an aid to number input.

A ‹scaled number› takes the form :

‹unscaled number›E[‹sign›]‹digits›

where ‹sign› is plus or minus, and ‹digits› is one or more decimal digits. The ‹sign› and ‹digits› are the exponent part. ‹White space› is allowed around the non-digit characters, but not within the digits portion. The value of the number is the value of the ‹unscaled number› multiplied by the power of 10 given by exponent part. (Note that the absolute value of the exponent part may not exceed 99.).

A ‹based number› may take any of the following forms :

&‹hexadecimal digits›
&H‹hexadecimal digits›
&X‹binary digits›

where ‹binary digits› are digits in the range 0..1, ‹hexadecimal digits› are digits in the range 0..9 and letters in the range A..F (or a..f). The ‹based number›'s decimal equivalent may not be greater than 65535. ‹White space› is allowed around the non-digit characters, but not within the digits portion. (‹Based number›s are treated as Integer, see 2.6 below.)

A ‹line number› takes the form:

‹digits›

where ‹digits› is at least one decimal digit. Line numbers are limited to the range 1..65535.

A ‹number› is terminated by any character which may not appear in a ‹number›.

2.5 String Constants.

A ‹string constant› is an arbitrary collection of characters enclosed in double quotes, or starting with double quotes and terminated by end of line (in which case trailing spaces do not form part of the string). Characters outside the normal ASCII printing range may appear in string constants, with the exception of Null (character value 0). A string constant may be up to 255 characters long.

2.6 Data Types.

BASIC handles two broad types of data, numeric and string.

Strings may be from 0 to 255 characters long. The characters in a string are represented by values in the range 0..255 (ie they are bytes). It is assumed that the ASCII character set is in use, but most string processing is independent of the values of the component characters of strings.

Numeric data may take one of two forms, Integer or Real :

Integer data is held as two byte two's complement numbers, so it can take values in the range −32768..+32767. In some instances the sixteen bit two's complement form is treated as an unsigned value, in the range 0..65535 - see section 2.10 below.

Real data is held in a five byte binary floating point format, with four bytes of mantissa and one byte of binary exponent. The largest absolute value representable is approximately 1.7E+38, the smallest (other than zero) is approximately 2.9E−39. The four byte mantissa affords a little over nine decimal digits of precision.

The type of a ‹number› is implied by its form :

‹Based number›s are always Integer, the unsigned value of which is mapped to its two's complement equivalent (unsigned values greater than 32767 map to 65536 − unsigned value).

‹Unscaled numbers› are treated as Integer if there is no decimal point and the value can be represented in the Integer range. Otherwise the number is treated as Real.

‹Scaled numbers› are Real.

2.7 Variables.

Variables have three attributes, name, type and organisation. A ‹variable name› takes the form ‹name›[‹type marker›]. The type of data referred to by a ‹variable name› is either given by the ‹type marker›, or is assumed to be the current default. The ‹type marker›s are:

%	for Integer
!	for Real
$	for String

The default type assumed depends on the first character of the ‹variable name›, and may be set dynamically by `DEFINT`, `DEFREAL` and `DEFSTR` commands. The initial default for all variables is Real.

Variables may be ‹simple variable›s which have a single value, or ‹array variable›s which are a collection of values of the same type. Arrays are described below.

Note that the same ‹name› may be used for variables of different types and organisations, and that these constitute different variables.

Variables do not need to be declared in BASIC, they are brought into being by their first use, with a value of zero or null string as appropriate.

2.8 Arrays.

BASIC supports arrays of all data types. An array is a collection of data items of the same type, each of which is referenced by the array variable name followed by a suitable number of subscripts, thus :

‹variable name› (‹subscripts›)

where ‹subscripts› is: list of: ‹integer expression›
and the ‹integer expression› yields a value in the correct range.
(Note that square brackets may be used instead of round brackets.)

The type of the data in the array is given by the type of the ‹variable name›.

An array may have any number of dimensions. Arrays may be declared explicitly in a `DIM` command, or implicitly by use. When an array is declared the number of dimensions and the upper bound on each is set (in an implicit declaration the upper bound taken is 10). Thereafter any reference to the array must use the same number of subscripts, and each one must be in range. It is not possible to change the dimensions of an array.

The lower bound on all array subscripts is 0.

2.9 Type Compatability and Conversion.

BASIC generally treats the numeric types as being compatible with each other. String type is only compatible with itself.

When dealing with numbers of different types BASIC will type convert automatically. Where there are no other constraints numbers are 'widened' to the 'larger' type involved, so Integers are widened to Real. Widening can never fail.

Numbers may also be forced to a given representation. Converting Real to Integer requires the number to be rounded to an integer, which must then lie in the Integer range of −32768..+32767, otherwise an Error 6 (`Overflow`) is generated.

2.9.1 Rounding.

There are several different ways in which a number in one representation may be rounded to another, smaller, representation. A given number X in the larger form, if it cannot be exactly represented in the smaller, will be bracketed most closely by two values X1 and X2 (which can be exactly represented), where X1 < X < X2. The common rounding schemes choose one of these values thus :

a. Truncation or Round Toward Zero

The excess digits are simply discarded: positive X −> X1,
negative X −> X2.

b. Round Toward −infinity

X −> X1

c. Round Toward +infinity

X −> X2

d. Round to Nearest.

X −> X1 or X2, depending on which is nearest to X.

Various schemes exist for choosing between X1 and X2 when X is exactly half way between the two. BASIC rounds away from zero in this case.

Unless otherwise stated, the term 'rounding' refers to Round to Nearest as defined above.

2.10 Unsigned Integer.

In some instances BASIC requires an Integer in the full unsigned 16 bit range of 0..65535 - for example PEEK requires a machine memory address. Floating point numbers are rounded to integer, which must then lie in the range −32768..+65535. The unsigned equivalent of the two's complement form of negative numbers is then used - that is to say negative numbers are treated as 65536+number.

2.11 Expressions.

There are four classes of expression: numeric, string, relational and logical. The precedence of operators is designed so that expressions work as might be expected. Where the order of evaluation is not forced either by the precedence rules or by brackets, evaluation proceeds from left to right.

2.11.1 Numeric Expressions.

The syntax of numeric expressions is :

‹numeric expression›	is: ‹numeric item›[‹operator›‹numeric item›]*
‹numeric item›	is: ‹unary minus›‹numeric item›
	or: ‹numeric constant›
	or: ‹numeric variable›
	or: ‹numeric function›
	or: (‹numeric expression›)
	or: (‹relational expression›)

Where ‹numeric variable› is a reference to a simple variable or an array item of either of the numeric types. A ‹numeric function› is a function returning either of the numeric types.

The ‹operator›s, in order of precedence, are :

↑	Exponentiation	Raises the value on the left to the power of the value on the right. Forces both values to Real before operation, and produces a Real result.
–	Unary minus	Notwithstanding the syntax, Unary minus has a lower precedence than Exponentiation.
*	Multiply	Real or Integer multiplication.
/	Divide	Real division. If either value is Integer it is forced to Real before operation.
\	Integer Division	Forces both values to Integer. The result is truncated to an Integer.
MOD	Integer Modulus	Forces both values to Integer. The result is the remainder after Integer arithmetic.
+	Addition	Real or Integer addition.
–	Subtract	Real or Integer subtraction.

Note that Exponentiation binds tightest and that Multiply and Divide have equal precedence, as do Addition and Subtraction. Except where the operator has particular requirements, the values on either side are forced to the same type by widening.

Division by zero generates an Error 11. If there is an ON ERROR GOTO set, then it is invoked in the usual way. If there is no ON ERROR GOTO set, then integer division by zero causes execution to cease; but real division by zero does not terminate execution, as the zero may be the result of underflow in a previous operation. Instead the 'Division by zero' message is generated, the largest representable number (approx. 1.7E+38) with appropriate sign is returned, and execution continues.

Integer division by zero generates an Error 11, in the usual way.

In the event of overflow when the arguments are Integer the values are widened to Real and the operation repeated.

In the event of overflow when the arguments are Reals an Error 6 ('Overflow') is generated. If there is an ON ERROR GOTO set, then it is invoked in the usual way. If there is no ON ERROR GOTO set, the 'Overflow' message is generated, the largest representable number (approx. 1.7E+38) with appropriate sign is returned, and execution continues.

2.11.2 String Expressions.

The syntax of string expressions is :

‹string expression›	is: ‹string item›[+‹string item›]*
‹string item›	is: ‹string variable›
	or: ‹string constant›
	or: ‹string function›
	or: (‹string expression›)

Where ‹string variable› is a reference to a simple variable or an array item of type string. A ‹string function› is a function returning a value of type string.

The effect of the + operator on strings is to concatenate them. A new string is produced which is the first string immediately followed by the second. If the resulting string would be longer than 255 characters, an Error 15 is generated.

2.11.3 Relational Expressions.

Relational expressions compare two numeric values or two string values. Thus :

‹relational expression›
is: ‹numeric expression›‹relation operator›‹numeric expression›
or: ‹string expression›‹relation operator›‹string expression›

Note that ‹relation operator›s have a lower precedence than any of the operators in either string or numeric expressions, and a higher precedence than any logical operator.

The ‹relation operator›s are :

<		Less than
<=	=<	Less than or Equal
=		Equal
>=	=>	Greater than or equal
>		Greater than
<>		Not equal

Where the arguments are numeric the types are made the same by the process of widening described in 2.9 above. The meaning of the relation is as might be expected.

Where the arguments are strings, the meaning of the relations requires some explanation:

Two strings are equal when they are the same length, and corresponding characters are the same.

One string is less than another if :

they are equal up to the end of the first string and the second string is longer

or: the first character which is not the same in the two strings is smaller in the first string. (Where the values representing the characters are being compared.)

BASIC does not support a Boolean type. Relational expressions yield an Integer value, −1 for True and 0 for False. Note that the IF and WHILE commands treat a value of 0 as False and any other value as True.

2.11.4 Logical Expressions.

BASIC does not support a Boolean type. Logical expressions perform bit-wise boolean operations on Integers. Logical expressions are :

‹argument›[‹logical operator›‹argument›]*

where:

‹argument›	is: NOT ‹argument›
	or: ‹numeric expression›
	or: ‹relational expression›
	or: (‹logical expression›)

The arguments for logical operators are forced to Integer, Error 6 results if an argument will not fit into the Integer range. The operation is then performed for each bit of the 16 bit two's complement representation of the Integer.

The NOT unary operator inverts each bit in the argument (0 becomes 1, and vice versa).

The dyadic operators, in order of precedence, and their effect on each bit are :

AND	Result is 0 unless both argument bits are 1
OR	Result is 1 unless both argument bits are 0
XOR	Result is 1 unless both argument bits are the same

The result of a relational expression is either −1 or 0. The representation for −1 has all bits of the Integer are 1; for 0 all bits of the Integer are 0. The result of a logical operation on two such arguments will yield either −1 or 0.

2.12 Functions.

Functions are subroutines which take a number of parameters, and which return a single value. Functions are invoked by their use as arguments in expressions, the value of the argument being the value returned by the function. There are two classes of function, built-in and user.

2.12.1 Built-in Functions.

These are part of the BASIC language, and are described in section 11. Unlike user functions the type of the result is not necessarily dictated by the type of the function name. For example the function ABS returns a result of the same type as its argument. Some built-in functions have optional parameters, and some allow different parameter types, neither of which is possible with user functions.

2.12.2 User Functions.

User functions are defined by the DEF FN command. This associates a name with a formal parameter list and an expression.

Function names take the form FN‹name›[‹type marker›], where the FN serves only to distinguish the name from variable names, and does not count toward the forty possible characters of the ‹name› part. As with variables, functions with the same name but different types are different functions.

The formal parameter list declares a number of variables local to the function. When the function is invoked actual parameters must be given which conform in number and type to the formal parameters - noting that BASIC will automatically change numeric types as required. The actual parameters are expressions, which are evaluated before the function is called, and whose values are copied to the formal parameters.

The body of the function is a single expression. It is not possible to have any form of control structure in a function. The expression may include references to other functions and to global variables as well as to the formal parameters - though, of course, a formal parameter with the same name as a global variable makes the global variable's value inaccessible.

When a function is evaluated the result of the expression is converted to the same type as the function name, and the converted value returned.

2.13 Decimal Fractions and Binary Floating Point.

It is a sad fact that few decimal fractions have an exact representation as binary floating point numbers. Floating point numbers are rounded before they are printed so the inexact representation may be obscured.

The effect may be seen when comparing values which are the result of separate sequences of arithmetic operations. The two values may appear equal when printed, but their binary representations are different. The same effect may be seen if two such numbers are subtracted and the result is not zero.

Unless decimal fractions are avoided altogether, by multiplying all numbers by suitable powers of ten (treating all money values as pence, for instance) it is impossible to avoid these effects. They can be reduced by rounding explicitly using the built in function ROUND, which will round numbers which print the same to the exactly the same value. Alternatively the slight differences may be retained, and account taken of their existence when numbers are compared or subtracted.

2.14 External Commands.

It is possible for commands to be directed at other software in the machine -either in memory or in expansion ROMs. These commands are distinguished from BASIC commands by prefixing the command name with a vertical bar ('|') When an external command is encountered BASIC attempts to find external software to claim it. If the command is claimed then BASIC interprets the rest of the statement as expressions, separated by commas. The expressions are evaluated, and passed to the external command subroutine as parameters.

As far as BASIC is concerned, external command names take the form of a BASIC ‹name›. The characters in the name are converted to upper case before external command tables are searched. The vertical bar is simply a marker to indicate to BASIC that an external command follows - it is not part of the external command name.

External commands may be of two types, 'Background' and 'Foreground'. A Background command returns to BASIC once it has finished - its purpose being to provide access to such extra facilities as lightpens, discs, clever printers and so on. A Foreground command does not return to BASIC, but takes over the machine completely - a self contained game or another language (such as Forth) are examples of Foreground commands.

3. Direct and Program Mode.

BASIC works in one of two major modes, Direct and Program. In Direct Mode BASIC takes lines typed at the keyboard and executes the commands when the terminating carriage return is entered. In Program Mode BASIC takes lines from the program currently in memory and executes the commands in each. With very few exceptions all commands may be used in either mode.

3.1 Direct Mode.

Direct Mode may be sub-divided into :

- Direct Commands
  In which commands are entered which are then immediately executed.

- Program Input
  In which new or replacement program lines are entered, or existing program lines are removed.

- Program Amendment
  In which existing program lines are altered.

Direct Commands mode is straightforward. It is possible to use variables in Direct Commands mode. Certain commands cause all variables to be lost, these include `RUN`, `NEW` and `LOAD`.

Program Input may be done simply by entering a line starting with a number. When the line is terminated (by **[ENTER]**) it is inserted into the current program, replacing any existing line with the same number. (Note that a line consisting solely of a line number deletes any existing line with that number.) The `AUTO` command causes BASIC to generate a sequence of line numbers for Program Input.

Program Amendment is supported by the `EDIT` command, see Section 6.

Note that BASIC does not perform any checking of program lines when they are entered or amended.

3.2 Program Mode.

In Program Mode BASIC automatically steps through the lines of the current program, obeying the commands it finds there. Lines are taken in line number order, except where explicit control structures dictate otherwise. Various commands cause BASIC to leave Program Mode and return to Direct Mode. The `END` command signals an expected end point of a program. There is an implied `END` command after the last line of a program.

Variables persist when BASIC returns from Program to Direct Mode, which may be useful when debugging a program.

Error Processing Mode is a sub-mode of Program Mode, and is entered when an error occurs while an ON ERROR GOTO is active.

An **ESC** from the keyboard while a program is running will cause BASIC to stop executing the current program and wait for the next keystroke. The action taken depends on the key pressed :

ESC BASIC will return to Direct Mode with a break message. Provided the program is not altered in any way, it may be restarted by CONT as if the Break had never happened.

SPACE BASIC resumes running the current program. The space is discarded.

Any other key causes BASIC to resume running the current program. The key is not discarded, and may be processed later.

4. The Screen.

The screen is a 'pixel screen', that is to say each dot, or picture element, may be set individually. Each pixel is represented by a number of bits in a fixed size area in main memory. The number of bits per pixel, and therefore the number of pixels on the screen, depends on the screen mode.

The screen is presented in two forms, a text screen and a graphics screen. The text screen software divides the pixel screen into a number of character cells and looks after the display of characters and symbols. The graphics screen software supports access to the individual pixels and includes facilities for line drawing and point plotting.

4.1 Modes, Resolution, Colours and Inks.

The screen operates in one of three modes. Each mode is capable of displaying a different number of pixels with a different number of colours used at the same time -where the more pixels, the fewer colours. The screen occupies 16K bytes of memory, with 4, 2 or 1 bits per pixel, depending on the mode.

The screen is capable of drawing pixels in 27 different colours (including white and black). The value given in each pixel is translated into one of the 27 colours by the hardware Palette. The pixel value is called an Ink number. The colour associated with each Ink may be changed at any time, instantaneously affecting the colour of all the pixels containing that Ink number. Two colours may be associated with an Ink, the colour in the Palette alternates between the two, giving a flashing effect.

Characters are formed on the screen in character cells eight pixels across by eight pixels deep.

The modes are as follows :

Mode 0: 25 lines of 20 characters or 160 dots across by 200 dots deep.
16 Inks (4 bits per pixel)

Mode 1: 25 lines of 40 characters or 320 dots across by 200 dots deep.
4 Inks (2 bits per pixel)

Mode 2: 25 lines of 80 characters or 640 dots across by 200 dots deep.
2 Inks (1 bit per pixel)

There is a border around the usable portion of the screen. The border has its own separate Ink entry in the Palette - so it may be set to a different colour (or alternating colour pair) to the rest of the sceen.

Colours are numbered from 0 to 26, colour 0 is Black, colour 26 is White - see Appendix VI for a list of the other colours. The numbering is arranged so that the colours are in grey scale order, which is of interest to those with monochrome screens.

4.2 The Text Screen and User Definable Characters.

When the screen is used to display text it is divided into character cells, each eight pixels wide by eight pixels deep. This gives 25 lines of characters in all modes, but a different number of characters per line - see above. Lines (the 'y' position) are numbered from 1 at the top of the screen, so the bottom line is line 25. Characters are numbered along the line (the 'x' position) from 1 at the left hand edge of the screen, so the rightmost character is character 20, 40 or 80 depending on the mode.

The character set comprises 256 different characters. The characters with values 32 to 126 are the usual ASCII characters. The other characters are set up to be various symbols and 'block graphics'. The appearance of each character is defined by an eight byte matrix definition, in which there is one bit for each of the eight by eight pixels in the character cell. The last 16 characters are implicitly 'user definable', which is to say that the matrix definition may be changed at will - though characters already displayed are not affected. The number of user definable characters may be altered using the `SYMBOL AFTER` command.

In general two inks are used when a character is written to the screen. Where a bit in the character's matrix is set to one the corresponding pixel in the character cell is set to the Pen Ink. Where a bit in the character's matrix is set to zero the corresponding pixel in the character cell is affected as follows:

in Normal or Write mode:	the pixel is set to the paper ink so that the character background is also filled in.
in Transparent mode:	the pixel remains unchanged so that the previous background shows through the character - hence the name *transparent*.

4.3 Screen Streams and Text Windows.

The first eight text streams, streams 0..7, send characters to the text screen. Each stream has a separate set of attributes :

- Window
- Paper Ink
- Pen Ink
- Cursor Position
- Transparent Option
- Text At Graphics Cursor Option

The area of the screen used by a given stream is known as its Window. The `WINDOW` command may be used to restrict a stream to some fraction of the screen. This windowing may be used to separate independent text streams, or to separate text and graphics.

Except where their windows overlap, the screen streams may be considered to be entirely separate text screens. Where windows overlap the contents of the overlapping areas reflect the latest activity in those areas.

The text cursor may be moved outside the current window, but before a character or the cursor blob is drawn, and before various control codes are obeyed, the cursor is forced to a legal window position, which may cause the window to roll up or down. See Appendix IV for more detail.

Characters sent to the text screen are interpreted as follows :

0.. 31 Control Characters.	These have special meaning to the text screen, as described in Appendix IV. Control characters may be used to move the cursor, clear the window, set inks and so on. To display the symbol associated with a character in the range 0..31 the character must be preceded by a CHR$(1).
32..127 ASCII Characters.	The symbol displayed may be altered using the User Definable Symbol mechanism.
128..255 Assorted Symbols.	The last 16 symbols are implicitly user definable, though this number may be changed (up or down) by the SYMBOL AFTER command.

4.4 Graphics Screen, Coordinate System, Window, and Origin.

There is only one graphics screen. The graphics screen has its own coordinate system and window.

The graphics screen coordinates divide the screen into 640 points across the screen by 400 points down it. These coordinates are mapped into screen pixels such that graphics is independent of screen mode, except for the variation in resolution. The horizontal mapping depends on screen mode :

Mode 0: 1 screen pixel corresponds to 4 points (low resolution)
Mode 1: 1 screen pixel corresponds to 2 points
Mode 2: 1 screen pixel corresponds to 1 point (high resolution)

The vertical mapping is not affected by the mode, 1 screen pixel corresponds to 2 points. (The vertical mapping arranges that one point vertically corresponds roughly to one point horizontally, giving an approximately 1:1 aspect ratio.)

The graphics screen coordinates grow up and to the right, and are based at zero. So the bottom left hand corner of the screen is at absolute position (0,0), and the top right hand corner of the screen is at absolute position (639,399). Graphics coordinates may take any Integer values, including negative values.

The coordinate system origin may be moved away from the bottom left hand corner of the screen by the ORIGIN command. When a graphics position is mapped onto a screen pixel the coordinates are truncated toward the origin. A change of origin is, therefore, almost but not quite, equivalent to adding an offset to coordinates.

The graphics window may use the whole screen, or be restricted to some part of it. The graphics window is set by optional parameters in the ORIGIN command. The graphics cursor may be moved outside the window but any attempt to draw outside the window is ignored.

Graphics and text may be mixed to a certain extent by the Text at Graphics Cursor facility. A text stream may be directed to display characters at the Graphics Cursor position. This allows characters to be placed anywhere on the screen, rather than only at character cell positions. Characters written at the graphics cursor are not affected by the transparent option (see above) ,but are affected by the Graphics Ink Mode (see below).

The way pixels are inked in by PLOT, DRAW or Text at Graphics Cursor (TAG) depends on the Graphics Ink Mode, which may be one of the following:

- Normal The pixel is set to the new ink.
- XOR The pixel is set to the XOR of the old and new inks.
- AND The pixel is set to the AND of the old and new inks.
- OR The pixel is set to the OR of the old and new inks.

4.5 Screen Commands and Functions Summary.

BORDER	- Set the Border Ink
CLG	- Clear Graphics Screen
CLS	- Clear Text Window for given stream
DRAW	- Draw line on Graphics Screen - to absolute position
DRAWR	- Draw line on Graphics Screen - to relative position
INK	- Set a given Ink
LOCATE	- Position text cursor for given stream
MODE	- Set new screen mode
MOVE	- Position graphics cursor - to absolute position
MOVER	- Position graphics cursor - to relative position
ORIGIN	- Set graphics coordinates origin and graphics window
PAPER	- Set Paper Ink for given stream
PEN	- Set Pen Ink for given stream
PLOT	- Plot point on graphics screen - at absolute position
PLOTR	- Plot point on graphics screen - at relative position
POS	- Return horizontal text cursor position for given stream
SPEED INK	- Set periods for flashing Inks
SYMBOL	- Set User Defined Character matrix
SYMBOL AFTER	- Change number of User Definable characters
TAG	- Set Text At Graphics Cursor mode for given stream
TAGOFF	- Turn off Text At Graphics Cursor mode for given stream
TEST	- Return Ink used at point on graphics screen - absolute
TESTR	- Return Ink used at point on graphics screen - relative
VPOS	- Return vertical text cursor position for given stream
WINDOW	- Set text window for given stream
WINDOW SWAP	- Exchange text screen streams
XPOS	- Return graphics X (horizontal) position
YPOS	- Return graphics Y (vertical) position

5. Keyboard and Joysticks.

The keyboard has three groups of keys, the main keyboard complete with various 'shift' keys, the numeric key pad and the cursor control keys. Two joysticks are supported, and are treated as part of the keyboard - joystick 0 as a separate part, joystick 1 overlapping some keys on the main keyboard. Up to thirty-two keys may be designated as Function Keys, each of which may be programmed to produce a string of characters.

Each key has three values associated with it, each in the range 0..255. The value generated when the key is pressed depends on the 'shift' state. The three 'shift' states are Normal, Shift and Control, and are produced :

Normal	when neither of the other 'shift' states is active.
Shift	when one of the **Shift** keys is held, or when Shift Locked, and **[CTRL]** is not pressed. Shift Lock is set by pressing **[CTRL]** & **[SHIFT]** together, and reset by pressing the same combination again.
Control	when the **[CTRL]** key is also pressed, irrespective of any other 'shift' state.

The **[CAPS LOCK]** key toggles the Caps Lock state. Caps Lock causes key values in the range 'a'..'z' to be mapped to the equivalent values in the range 'A'..'Z'. (So Caps Lock is not equivalent to Shift for the alphabetic keys, which is of interest when changing the keyboard.)

The **[ESC]** key is used to interrupt BASIC. When BASIC is not running a program, but is otherwise active - for example during EDIT - pressing **[ESC]** abandons the operation and returns to Direct Mode. When BASIC is running a program there is a two stage action, when **[ESC]** is pressed BASIC suspends program execution and waits for the next key, whose effect is :

[ESC]	stops BASIC and returns to Direct Mode.
[SPACE]	continues running. The space character is discarded.
other	continues running. The character is retained for processing later.

The **[ESC]** key in combination with **[SHIFT]** and **[CTRL]** resets the machine, leaving it in the same state as when it was first turned on (provided that **[ESC][SHIFT]** and **[CTRL]** are the only keys held down).

Keys may be set to repeat when pressed and held down. When a key is first pressed it generates its key value. If the key is set to repeat then after it has been held down for a given period (known as the start-up delay) the key value is generated again. If the key remains down then the key will repeat again at a rate determined by the repeat period, which is generally smaller than the start-up delay. The initial settings are 0.6 sec. start-up delay and 0.04 sec. repeat period (ie. 25 characters per second).

The keyboard is scanned fifty times a second. The scan compares the current and previous states of the keyboard to keep track of key changes. A buffer of keys pressed is maintained.

The repeat setting and the three values associated with each key may be changed at any time. The keyboard may, therefore, be tailored to suit a particular application. (It is not, however, effective to try to change the **[SHIFT]** or **[ESC]** keys.) The value generated by a key may be :

0..31	These values tend to have special effects when printed, but are treated as ordinary characters coming from the keyboard except for &0D (13) which is Carriage Return.
32..127	Ordinary characters, usually interpreted as ASCII
128..159	Special characters which are expanded according to the strings set by KEY commands, see below.
160..223	Ordinary characters.
224..254	Special values used for edit and copy cursor keys.
255	Ignored

The thirty two special characters (called expansion tokens), in the range 128..159, may be used to implement Function Keys - keys which generate more than one character. The keyboard software maintains a string associated with each of the expansion tokens. When one of these is read from the keyboard buffer it is replaced by its associated string. The strings may be changed at any time. The initial state of the keyboard has the numeric keypad set to the characters 128..139, and their associated strings set according to the keytops. **[CTRL][ENTER]** is set to the character 140 and the string RUN"‹carriage return›. The other nineteen expansion tokens are not used in the initial keyboard state.

Summary of Keyboard Commands and Functions:

INKEY	- Test current state of given key.
INKEY$	- Fetch next key from keyboard interrupt buffer.
JOY	- Test state of given joystick.
KEY	- Set new 'expansion token' string.
KEY DEF	- Redefine key on keyboard.
SPEED KEY	- set keyboard start-up delay and repeat period

6. Line Input, Edit and Copy.

The machine includes a line editor and the ability to copy text from the current window into the current line. These facilities are available whenever BASIC is expecting a line from the keyboard.

The line editing facilities allow the cursor to be moved within the current line, and for characters to be removed or inserted at the cursor position. When the line is terminated, by **[ENTER]**, the result is passed to BASIC, including any text to the right of the cursor (if the cursor has been moved back from the end of the line). Pressing **[ESC]** during line input or edit abandons the line.

The `EDIT` command takes a specified line, makes it the current line, displays it and sets the cursor to the first character. The line may then be changed. When the line is terminated BASIC inserts it back into the program. Note that changing the line number does not move the line, but creates a new line (or replaces another existing line).

The copy facilities allow a copy cursor, which is separate from the edit cursor, to be moved around the screen. Each time the **[COPY]** key is pressed the character under the copy cursor is copied and inserted at the edit cursor position, the copy cursor then moves forward.

The cursor keys have three meanings, depending on 'shift' as follows :

'Normal'	move the Edit cursor around the current line. Note that moving the cursor beyond either end of the current line is not allowed. While the current line is empty these keys may be used to move the cursor around the screen.
'Shifted'	- move the Copy cursor around the screen.
'Control'	**[CTRL]**-Up - move to the start of the current logical line. **[CTRL]**-Down - move to the end of the current logical line. **[CTRL]**-Left - move to the start of the physical line. **[CTRL]**-Right - move to the end of the physical line.

The other keys of importance while line editing are :

[DEL]	Delete the character to the left of the edit cursor. Moves the cursor and the rest of the line one character to the left.
[CLR]	Delete the character under the edit cursor. Moves the rest of the line one character to the left.
[ESC]	Abandon the current line.
[CTRL]-[TAB]	Toggles the Insert/Overstrike mode.
[ENTER]	Finish editing the line and issue it to BASIC.

Insert mode is set at the beginning of line input or edit. New text may be inserted into the current line simply by typing it in, the rest of the line moves one character to the right to make room for the inserts. **[CTRL] - [TAB]** toggles the mode between Insert and Overstrike. In Overstrike mode new text replaces existing characters in the current line.

7. Streams.

All BASIC input and output other than graphics uses 'Streams'. There are ten output streams and ten input streams. The output streams are :

0..7	Text screen streams
8	Printer stream
9	Cassette output file

The input streams are :

0..7	Keyboard, with prompts sent to corresponding screen stream
8	Keyboard, with prompts sent to printer stream
9	Cassette input file

7.1 Output Streams.

Text output commands, such as PRINT, include an optional stream parameter (if the parameter is omitted, stream #0 is assumed). This makes the commands consistent across the different devices. Since the stream number in the parameter is given by an expression, it is possible to write a program which is to output to printer or cassette but test it using output to the screen.

The screen streams each have their own Window, Pen, Paper etc. attributes. The commands associated with setting these do not have Printer or Cassette equivalents.

Before writing to the cassette stream an OPENOUT command must be issued to set the name for the new file. When all required data has been written a CLOSEOUT command is required to flush any buffered data and terminate the file on tape.

7.2 Input Streams.

There are only two physical sources of input to BASIC, the keyboard and the cassette. When an INPUT command is executed BASIC issues a prompt before awaiting input. The keyboard streams all take input from the keyboard, but the prompt is directed at the corresponding output stream. In the case of cassette input prompts are ignored.

Before reading from the cassette an OPENIN command must be issued to set the name of the input file. When all required data has been read a CLOSEIN command is required to set BASIC back to a known state. The EOF function may be used to test for end of file on the cassette input.

7.3 Cassette Files.

The cassette input and output streams are refered to as the file streams. It is possible to read data from one cassette and write to another 'simultaneously'. Since there is only one cassette player on the system this requires cassettes to be changed at the appropriate moments.

BASIC reads and writes files in 2K byte blocks. When a file is opened for input (using `OPENIN`) the tape is searched for the required file, and the first block read into memory, ready for processing. When a file is opened for output (using `OPENOUT`) nothing is written to tape until the 2K + 1th byte is written and BASIC is forced to empty its buffer.

Programs reading and writing cassette files should close them (using `CLOSEIN` and `CLOSEOUT`) when they are finished with. `END` or end of program implicitly close any cassette files. When an output file is closed any buffered data is written to tape.

Cassette files are abandoned by `NEW`, `SAVE`, `MERGE`, `LOAD`, `RUN`, `CHAIN`, `CHAIN MERGE` and `CLEAR` commands. Abandoning an output file does not cause any buffered data to be written, so it will be lost.

8. Sound Generation.

The sound generation hardware has three independent channels (A, B and C). Each channel is capable of producing square wave tones in the approximate range 30..125k Hertz and/or pseudo random noise. The volume of each channel may be set to one of sixteen levels (including off). The commands available provide for simple sounds with constant volume, tone and noise, as well as for more complicated sounds in which the volume and/or tone vary during the sound under the control of 'envelopes'. Facilities also exist to ensure that sounds sent to separate channels are in step when required.

The sound generation hardware does not work with frequencies but with periods. Tone settings are expressed as tone periods, where a period P produces a tone of frequency F such that :

F = 125000/P

(and conversely a period P is required for a frequency F where P = 125000/F). The tone enveloping allows the tone period to be varied during the generation of a sound. See Appendix VII for a list of recommended tone periods corresponding to the usual even tempered scale.

The noise generator produces a tone whose frequency varies at random around a given value. The noise setting is also expressed as a period, where the frequency and period have the same relationship as tone frequency and tone period.

While each channel may be set to produce tone and/or noise, the hardware has only one noise period setting, so it is not possible to have different frequency noise on different channels at the same time.

The duration of a sound may be set explicitly, or may be governed by the length of a volume envelope. If set explicitly the sound stops after the given time, irrespective of any volume or tone envelopes. If the duration is set by the length of the volume envelope, then the sound stops when the volume envelope is exhausted, irrespective of any tone envelope. It is possible to set a sound to repeat the volume envelope a given number of times - the sound will last until the end of the final repetition.

A volume envelope may be used to vary the volume of a sound. Each envelope may have up to five sections, each section being an incremental or an absolute change to the volume or, exceptionally, a hardware envelope setting:

An incremental section is specified as a number of steps, the size of each step (which may be negative) and a pause after each step (measured in 100ths of a second).

An absolute section is specified as a new volume and a pause before the next section (measured in 100ths of a second).

The hardware envelope setting may be used to take advantage of the sound chip's enveloping features where suitable.

A tone envelope may be used to vary the tone period of a sound. Each envelope may have up to five sections, each section being an incremental or an absolute change to the tone period:

> An incremental section is specified as a number of steps, the size of each step (which may be negative) and a pause after each step (measured in 100ths of a second).
>
> An absolute section is specified as a new tone period and a pause before the next section (measured in 100ths of a second).

A tone envelope may be set to repeat itself.

The enveloping facilities may be used to produce sounds which resemble various musical instruments, particularly when one channel is used for the basic tone and the others are used to add harmonics.

Each channel may have one sound active and up to four sounds queued awaiting processing. When an active sound terminates the system attempts to execute the next sound from the queue for the appropriate channel. The sound will start unless it is Held or awaiting a Rendezvous:

> When a sound is Held it remains waiting to be executed, and the channel produces no sound, until the channel is Released. The Hold/Release mechanism allows sound queues to be 'primed' by setting the first sound for each channel to be Held, adding further sounds to the queues, and then Releasing the channels.
>
> The Rendezvous mechanism may be used to force sounds for two or all channels to start together. When a sound is placed in the queue for a channel it may be set to Rendezvous with a sound in one or both of the other channels. When the sound comes to be executed it will be delayed until the other channels have sounds ready to be executed with reciprocal Rendezvous settings.

A sound may be issued with Flush enabled. The Flush terminates any active sound immediately and discards any queued sounds, leaving the queue empty and the channel inactive, ready for the new sound. The bleep sound produced by the Bell character performs a Flush before it is issued.

The state of the sound queue for a given channel may be tested. The system will report how many free entries there are in the queue and whether a sound is active or the channel is Held or awaiting a Rendezvous.

To facilitate the generation of sounds 'in background' each channel may be set to invoke a given BASIC subroutine when its queue is not full. The subroutine may issue a new sound for that channel. See Section 9 for further discussion of 'interrupt' subroutines.

Sound generation continues after `END` or end of program until the sound queues empty, but sound queue interrupts are disabled.

Suspending or stopping a program while sounds are active causes the current sounds to be suspended. When program execution is resumed sounds are also resumed where they left off. If a sound is issued while in Direct Mode any suspended sounds also resume.

Use of the cassette terminates all current sounds and clears all sound queues.

Summary of Sound Commands and Functions:

ENT	- Set Tone Envelope.
ENV	- Set Volume Envelope.
ON SQ	- Set Sound Queue interrupt.
RELEASE	- Release given sound queues.
SOUND	- Insert sound into given sound queues.
SQ	- Test state of given sound queue.

9. BASIC Interrupts.

Events external to BASIC may be used to trigger the calling of a BASIC subroutine. After each statement BASIC looks to see if any of the nominated events has occurred. When such an event is seen the subroutine associated with it is called, just as if there was a GOSUB following the previous statement. The events which may cause this Interrupt action are :

- A set of four timers.
- The three sound channel queues.
- Break **[ESC]/[ESC]**.

If there is more than one event outstanding the event with the highest priority takes precedence. While the event's subroutine is being executed it will not itself be interrupted except by an event of a higher priority. The priorities, in descending order, are :

Break **[ESC]/[ESC]**
Timer 3
Timer 2 and the three sound queues (all equal priority).
Timer 1
Timer 0

Care must be exercised in the use of Interrupts. Since an interrupt may happen at any time between statements the programmer must be careful to ensure that the interrupt subroutine is not adversely affected by an intermediate state of variables. Conversely the interrupt subroutine must be written to avoid unwanted interactions with the rest of the program, caused, for example, by the second use of a variable, or the effect of moving screen cursor positions.

The priority mechanism provides a measure of mutual exclusion. It is also possible to explicitly disable all interrupts (except Break interrupts), and to enable them again. Where interrupt subroutines interact with each other, or with the main program, interrupts may be disabled to ensure that a sequence of statements is executed indivisibly.

9.1 The Timers.

The four delay timers are independent, and have different priorities. The timers count down a given number of 1/50ths of a second and then interrupt. It is possible to set a timer to interrupt once or to reset the count after every interrupt, giving a regular repeated interrupt.

Since interrupts are not processed except at the end of statements the subroutine may not run for a little while after the timer count has expired. The repeating timer's count is reset immediately the previous count expires, so the interrupts are generated regularly even if there are long delays before the subroutine is entered. The system keeps track of up to 127 interrupts, so it is unlikely that any will be lost.

The timers continue to count if the program is suspended (by **[ESC]**) and if the program is stopped for any reason. If program execution is resumed any outstanding timer interrupts will be processed as soon as possible.

The `REMAIN` function returns the unexpired count from a given delay timer, and disables it. The single interrupt timer setting disables itself automatically when it interrupts.

9.2 The Sound Queue Interrupts.

The three sound queues have independent interrupts, but they all have the same priority. Once a sound interrupt subroutine has started, therefore, it will not be interrupted by another sound interrupt - so sound interrupt subroutines may share variables.

When a sound queue's interrupt is enabled it will immediately interrupt if the queue for that channel is not full, otherwise it will interrupt when the next sound starts and there is room for one more in the queue. The action of interrupting disables the event, so the subroutine must re-enable it if further interrupts are required. Attempting to issue a sound or testing the queue status will also disable any event enabled for queue or queues involved.

9.3 The Break ([ESC][ESC]) Interrupt.

This interrupt, when enabled, is triggered by the **[ESC][ESC]** pair (which otherwise stops program execution). This facility may be used to prevent **[ESC][ESC]** from stopping a program at all, or to perform some tidying up before stopping, or to implement some special feature.

9.4 Summary of Interrupt Commands and Functions.

`AFTER`	- Set timer to interrupt once after the given time.
`DI`	- Disable interrupts (except Break).
`EI`	- Enable interrupts.
`EVERY`	- Set timer to interrupt every 'n ticks'.
`ON BREAK GOSUB`	- Set Break interrupt subroutine.
`ON BREAK STOP`	- Disable Break interrupt.
`ON SQ GOSUB`	- Set Sound Queue interrupt subroutine.
`REMAIN`	- Return timer's unexpired count and disarm it.

10. Overview of Commands.

This section is not intended to describe the commands in detail, that is done in section 11. Here the commands are described in groups, in order to give some idea of their purpose and to place them in context.

10.1 Creation of Programs.

AUTO, DELETE, EDIT, LIST, MERGE, NEW, RENUM, SAVE

The commands in this group are all useful when preparing a new program, or amending an existing one.

AUTO is an aid to the entry of a new section of program. It generates line numbers as the new program lines are entered.

DELETE removes lines of program en masse.

EDIT allows existing lines to be amended.

LIST produces listings of all or part of a program to the screen, the printer or the cassette (ie. to a given stream).

MERGE takes a program from a file and adds it to the program in memory. This may be used, for example, to add useful subroutines saved on cassette to the current program. MERGE will accept a BASIC program file or an ASCII text file - it will not accept any other file types, in particular it will not accept a protected program file.

NEW clears the program area, disposes of all variables and generally tidies up, ready for a new program to be entered.

RENUM allows all or part of a program to be renumbered. References to lines which are renumbered are updated.

SAVE writes the current program, or the contents of an area of memory, to a cassette file. The program may be written in the internal form, in protected internal form, or as an ASCII file. Writing out the contents of an area of memory creates a Binary file, which may, for example, be a machine code program.

10.2 Loading and Running of Programs.

CLEAR, CHAIN, CHAIN MERGE, LOAD, MEMORY, HIMEM, RUN, FRE

CLEAR clears away all variables, closes cassette files and generally resets BASIC, except that the current program is retained.

MEMORY allows the memory available to BASIC to be changed by setting the address of the highest byte in memory which may be used by BASIC.

The HIMEM function returns the current highest byte in memory used by BASIC.

The FRE function returns how much memory within BASIC's limits is currently unused.

CHAIN and CHAIN MERGE enable one program to load and enter another. With CHAIN MERGE all or part of the current program may be retained. Variables are retained past CHAIN or CHAIN MERGE.

LOAD discards the current program and variables, resets all other status and loads a new program from cassette - BASIC program files or ASCII files may be loaded. LOADing a protected BASIC program is futile, since it will be immediately deleted! LOAD may also be used to read a Binary file into memory.

One form of the RUN command starts executing the current program, at the beginning or at a given line number. The other form of the command causes a new program to be loaded from cassette and execution to start from the beginning -BASIC program files or ASCII files may be loaded. RUN may also be used to load and execute a machine code program.

10.3 Program Termination.

END

Programs stop and return to Direct Mode after the last line has been executed. END may be used to terminate the program at some other point, or points.

10.4 Control Structures.

FOR, GOSUB, GOTO, IF, ON x GOSUB, ON x GOTO, RETURN, WHILE

BASIC supports two forms of loop, FOR and WHILE. A FOR loop has a control variable associated with it, which steps through a range of values - one per iteration - until a given end value is reached. The size of each step may be specified, including negative steps. The BASIC FOR loop is skipped if the initial value of the control variable exceeds the end value. NEXT marks the end of a FOR loop. A WHILE loop repeats until the given condition becomes false. The BASIC WHILE loop is skipped if the condition is false the first time. WEND marks the end of a WHILE loop. FOR and WHILE loops may be nested.

IF commands may be used to choose alternative or optional actions. IF is followed by an expression. If the expression yields a non-zero value the THEN part is chosen, otherwise if it is present the ELSE part is chosen. Both the THEN and ELSE parts may contain more than one command, but they must all be on the same line with the IF command. IF commands may be nested on the same line.

Simple subroutines are supported. GOSUB calls the subroutine that starts at the line given. RETURN terminates the subroutine, and returns control to the next command after the GOSUB which invoked it. All variables in BASIC are global. Subroutines may recurse.

GOTO causes an unconditional branch to the line given.

ON x GOSUB and ON x GOTO commands evaluate the expression x, and use the result to choose one out of a list of lines to call or jump to.

10.5 Variables.

DEFINT, DEFREAL, DEFSTR, DIM, ERASE

Where a variable or function name is given without an explicit type marker BASIC must assume a type. The type assumed depends on the first character of the name (which is an alphabetic character). DEFINT, DEFREAL and DEFSTR commands may be used to define the default type for each letter. Note that these default settings may be changed at will, but that the potential for confusion is enormous.

Unless otherwise explicitly declared in a DIM command the maximum for each subscript of an array is set to 10 by the first use of the array (implicit declaration). The minimum for each subscript of an array is zero.

Arrays may occupy large areas of memory. The ERASE command may be used to reclaim the memory occupied by arrays which are no longer in use.

10.6 Screen.

MODE, BORDER, INK, SPEED INK

MODE sets the screen into one of the three possible modes (see Section 4). Changing screen mode clears the screen and resets all text windows, text cursors, and the graphics window, origin and cursor.

BORDER sets the colour, or colours, to be used for the border around the active area of the screen.

INK sets the colour, or colours, to be used for the given Ink (see Section 4).

When an Ink is set to two colours the colour displayed alternates between the two. SPEED INK sets the rate at which this occurs.

10.7 Text Output.

PRINT, WRITE, ZONE, POS, VPOS

PRINT outputs numbers and strings to the given stream. Numbers are output 'free format'. PRINT divides the output into 'print zones' which impose some order on it. The ZONE command may be used to change the size of print zones. In a PRINT command the keyword USING introduces a format template, which controls the output of all further numbers and strings, without reference to print zones.

WRITE outputs numbers and strings to the given stream, inserting commas between each item and enclosing strings in double quotes. Numbers are output in 'free format'. WRITE is most useful for outputting data to cassette so that it may be read back again using INPUT.

The POS and VPOS functions return the current horizontal and vertical position on the given stream.

10.7.1 Screen Streams.

CLS, LOCATE, PAPER, PEN, TAG, TAGOFF, WINDOW,
WINDOW SWAP

These commands all take a stream parameter restricted to one of the eight screen streams - since they set screen specific attributes.

CLS clears the window for the given stream to its current Paper colour.

LOCATE moves the stream's cursor to a given position within its current window.

PAPER sets which Ink is to be used for the background pixels for all subsequent characters written to the given stream.

PEN sets which Ink is to be used for the foreground pixels for all subsequent characters written to the given stream.

TAG directs all further characters sent to the given stream to the graphics cursor position. This allows characters to be displayed at arbitrary positions on the screen.

TAGOFF turns off the TAG option.

WINDOW sets the screen position of the window for the given stream.

WINDOW SWAP exchanges all the attributes of two screen streams. When one of the streams is stream #0 (the default/system stream) this has the effect of moving further system output to a new window on the screen.

10.7.2 The Cassette Stream.

CLOSEOUT, OPENOUT

Before text may be written to the cassette stream an OPENOUT command must be issued to prepare the cassette handling and to specify the name for the file to be written. When all text has been written the cassette file must be closed using the CLOSEOUT command.

10.8 Text Input.

INPUT, LINE INPUT

These commands take a stream parameter. Apart from the cassette input stream all input is from the keyboard, the stream in fact specifies which output stream to use if a prompt is required.

INPUT fetches a line from the given stream, and may interpret parts of it as numeric input and parts as plain text, depending on the parameters of the INPUT command.

LINE INPUT fetches a line from the given stream without interpretation (but with line imaging) and assigns it to a string variable.

10.8.1 Cassette Input.

CLOSEIN, EOF, OPENIN

Before data is read from a cassette file the file must be opened using an OPENIN command, specifying the name of the required file. When the file is no longer required it must be closed using a CLOSEIN command.

It is an error to attempt to read beyond the end of a cassette file. The EOF function may be used to test whether there are further characters to be read.

10.9 The Cassette in General.

CAT, SPEED WRITE

CAT starts reading the cassette and reports the files it finds as it comes across them. CAT verifies that the files it finds are readable.

SPEED WRITE changes the speed at which data is written to cassette, the lower the speed the less the probability of errors, particularly when writing files to be read on other machines. The write speed has no effect on the reading of tapes, which adjusts itself to the tape as it is read.

10.10 Constant Data.

DATA, READ, RESTORE

All the DATA commands in a program, taken in line number order, make up a sort of file. This 'file' may be read using READ commands. The position within this 'file' may be reset by RESTORE commands.

10.11 Graphics Screen.

CLG, DRAW, DRAWR, MOVE, MOVER, ORIGIN, PLOT, PLOTR, TAG, TAGOFF, TEST, TESTR, XPOS, YPOS

CLG clears the graphics screen.

ORIGIN sets the position on the screen of the origin for the graphics coordinates. Optional parameters may also specify the graphics window, restricting the effect of graphics commands within an area of the screen.

DRAW and DRAWR draw lines on the graphics screen from the current graphics cursor position to, and including, a specified point. DRAW specifies the end point absolutely. DRAWR specifies the end point relative to the current graphics cursor position.

MOVE and MOVER move the graphics cursor to a new position. MOVE specifies the new position absolutely. MOVER specifies the new position relative to the current graphics cursor position.

PLOT and PLOTR move the graphics cursor to a given point and colour it in. PLOT specifies the point absolutely. PLOTR specifies the point relative to the current graphics position.

The TEST and TESTR functions move the graphics cursor to a given point and test what ink is used at that point. TEST specifies the point absolutely. TESTR specifies the point relative to the current graphics position.

TAG causes text sent to the given text stream to be displayed at the graphics cursor position, rather than at the usual text cursor position. TAGOFF reverses the effect of TAG.

The XPOS and YPOS functions return the current graphics x and y position, respectively.

10.12 Sound Generation.

ENT, ENV, ON SQ, RELEASE, SOUND, SQ

ENT and ENV set new values for Tone and Volume envelopes, respectively. Fifteen Tone and fifteen Volume envelopes are available.

SOUND attempts to insert a new sound into the sound queue for each of the channels specified in the command. SOUND will wait until there is room for the sound in all the queues required. SOUND disables any 'ON SQ' set for the channels specified.

The SQ function returns a value giving the state of the given channel's queue - the value indicates whether there is an active sound, how many free slots there are in the queue and the Hold and Rendezvous setting of the next sound to be executed. The SQ function disables any 'ON SQ' set for the channel.

Sounds may be Held waiting to be executed. The RELEASE command releases the next sound to be executed for the channels specified.

ON SQ sets an interrupt subroutine for the given channel. The interrupt subroutine is invoked immediately if the queue is not full, otherwise it is invoked when a slot becomes available. The 'ON SQ' is disabled when it interrupts or by a SOUND command or an SQ function.

10.13 Timer Interrupts.

AFTER, EVERY, REMAIN

AFTER enables the given delay timer to interrupt after a given period has passed. The timer is disabled when it interrupts.

EVERY enables the given delay timer to interrupt every time a given period has passed. The timer continues enabled until explicitly disabled.

The REMAIN function returns what time is outstanding before the given delay timer would interrupt, and disables the timer.

10.14 Enabling and Disabling Interrupts

EI, DI

DI disables interrupts. EI enables interrupts. DI and EI should be used where it necessary to execute a sequence of statements indivisibly.

10.15 Keyboard and Joystick.

KEY, KEY DEF, INKEY, INKEY$, JOY, SPEED KEY

A KEY DEF command changes the repeat state and the value generated by a key on the keyboard (or a joystick switch) in each of the 'shifts'. The keyboard may be completely reconfigured using KEY DEF commands.

A KEY command changes the string associated with one of the keyboard expansion tokens. There are thirty-two expansion tokens, which are intended for 'function' keys. The numeric keypad is set up to the first twelve expansion tokens. KEY commands may, therefore, be used to program the function keys.

If there is a character outstanding in the keyboard buffer INKEY$ returns it, otherwise INKEY$ returns an empty string.

The INKEY function directly tests the state of a given key.

The JOY function directly tests the state of the given joystick.

A SPEED KEY command sets the start-up delay and the repeat period for the keyboard.

10.16 User Defined Characters.

SYMBOL, SYMBOL AFTER

The representation of a character on the screen (the character matrix) may be changed using the SYMBOL command. The SYMBOL command may only refer to characters which have been declared as modifiable by a SYMBOL AFTER command. Initially the last sixteen characters are implicitly modifiable.

Matrices for all 256 characters are held, in ROM. To change a character's matrix it must first be copied into RAM. The SYMBOL AFTER command declares that the matrices for all characters above a given value are to be moved into RAM. The SYMBOL AFTER command allocates an area of memory by changing HIMEM, and copies the matrices for the nominated characters into it.

10.17 Arithmetic Functions

ATS, ATN, COS, DEG, EXP, FIX, INT, LOG, LOG10, MAX, MIN, PI, RAD, RANDOMIZE, RND, ROUND, SGN, SIN, SQR, TAN

ATN, COS, SIN and TAN are the usual trig functions (ATN is Arctangent). By default these expect angles to be expressed in radians. After a DEG command the functions work in degrees - so COS, SIN and TAN take angles expressed in degrees, and ATN produces a result in degrees. The RAD command sets the functions back to working in radians. PI is a 'function' returning the closest possible machine representation of pi.

EXP returns the value of e raised to the given power. LOG returns the natural log (base e) of its argument. LOG10 returns the log to the base 10. SQR returns the square root of its argument.

RND returns the next number in a pseudo random sequence. RANDOMIZE sets a new seed for the sequence - RANDOMIZE TIME is an interesting method of seeding.

MIN takes a variable number of arguments and returns the value of the smallest. MAX similarly returns the largest value.

FIX and INT return integral values. FIX rounds its argument toward zero. INT rounds its argument toward–Infinity. ROUND rounds its argument to a given number of decimal places, rounding to nearest. (See Section 2.9.1 for a discussion of rounding.)

ABS returns the absolute value of its argument. SGN returns the sign of its argument.

10.18 String Functions.

BIN$, DEC$, HEX$, INSTR, LEFT$, LEN, LOWER$, MID$, RIGHT$, SPACE$, STR$, STRING$, UPPER$, VAL

LEFT$, RIGHT$ and MID$ functions return part of the argument string. LEFT$ returns a given number of characters counting from the left end of the string. RIGHT$ returns a number of characters counting from the right. MID$ returns some portion taken from a given position in the string.

MID$ may be assigned to, so that a portion of its argument string is replaced by another.

INSTR searches one string for the first occurence of a given substring.

LEN returns the length of its argument string.

LOWER$ returns a copy of its argument, with any upper case alphabetic characters replaced by their lower case equivalents. UPPER$ similarly replaces lower case by upper case.

STRING$ returns a string consisting of a given character repeated a given number of times. SPACE$ returns a string consisting of a given number of spaces.

BIN$, DEC$, HEX$ and STR$ produce string representations of the value of their numeric arguments. BIN$ produces a string of binary digits, HEX$ a string of hexadecimal digits. STR$ produces a free format decimal string. DEC$ produces a formatted decimal string - with format controls similar to PRINT USING. VAL takes a string and, interpreting it as a number in much the same way as INPUT, returns its value.

10.19 Type Conversion Functions.

ASC, CHR$, CINT, CREAL, UNT

ASC takes the first character of a string and returns its integer value.

CHR$ takes an integer value and returns a single character string.

CINT takes a numeric value and rounds it to Integer, rounding to nearest. The result must be in the range -32768..32767.

CREAL takes a numeric value and converts it to Real.

UNT takes a numeric value and rounds it to Unsigned Integer, rounding to nearest.

10.20 Machine Level Operations.

CALL, INP, OUT, PEEK, POKE, WAIT

CALL invokes a machine code routine at a given address, passing a number of parameters (by value). The @ 'operator' may be used to pass the address of a variable's value. See Appendix II for a more detailed description.

PEEK and POKE read and write bytes of machine memory.

INP, OUT and WAIT give access to machine I/O space.

10.21 Error Trapping.

ERL, ERR, ERROR, ON ERROR GOTO, RESUME, ON BREAK

When BASIC detects an error its default action is to issue an error message and return to Direct Mode. This action may be overridden by an ON ERROR GOTO command, which sets the line number of a routine to process errors. The error processing routine may use the ERR function to establish which error has occurred and the ERL function to find where it occurred. The RESUME command may be used to return to normal running.

ON BREAK allows the program to trap the **[ESC]/[ESC]** program halt key sequence, and take alternative action.

10.22 Program Development.

CONT, STOP, TRON, TROFF

The STOP command halts a program in such a way that it may be restarted by CONT (provided the program is not altered in the meantime). STOP commands may be inserted in a program to halt it at interesting points so that the state of the program and variables may be examined.

TRON and TROFF turn on and off a tracing mechanism. When tracing is enabled, before each line is executed its number is printed on the console, enclosed in square brackets. This provides a simple trace of the program's execution.

11. The Commands and Built In Functions.

This section contains a detailed description of all Commands and Built In Functions, and is arranged in alphabetical order of keyword.

The following terms are used throughout the section :

‹address expression›

A ‹numeric expression›, which when rounded to an integer yields a value in the range -32768..65535. Negative values are then converted to their unsigned equivalent (by adding 65536).

‹array variable name›

The name of an array, which may include its ‹type marker›.

‹colour›

An ‹integer expression› yielding a value in the range 0..26, specifying one of the 27 possible colours (see Appendix VI).

‹expression›

An expression yielding a value of any type.

‹ink›

An ‹integer expression› yielding a value in the range 0..15, specifying one of the 16 possible palette entries.

‹integer expression›

A ‹numeric expression›, which when rounded to an integer yields a value in the range -32768..32767.

‹line number›

A decimal number in the range 1..65535.

‹line number range›

A ‹line number range› specifies all lines whose numbers are within the inclusive range given. The range may take one of the forms:

‹line number›	Only the given number is in the range.
‹line number› –	Lines from the given line to the end of the program.
–‹line number›	Lines from the beginning of the program to the given line.

‹line number›–‹line number›

Lines from the first ‹line number› to the second ‹line number›

‹logical expression›

BASIC does not support a separate boolean type. Integer value Ø is treated as false, and any other Integer value as True. A ‹logical expression› is, therefore, an ‹integer expression› by another name.

‹masked ink›

An ‹integer expression› yielding a value in the range 0..15, specifying one of the 16 possible inks. The value is then masked to force it into range for the current mode, thus :

Mode 0:	Mask = 15, final range = 0..15
Mode 1:	Mask = 3, final range = 0..3
Mode 2:	Mask = 1, final range = 0..1

‹numeric constant›

See Section 2.4.

‹numeric expression›

An expression which returns a Real or an Integer value, including expressions previously categorised as relational and logical.

‹numeric variable›

A reference to a variable, which may be an array item, whose type is Real or Integer.

‹quoted string›

A ‹quoted string› is between 0 and 255 characters enclosed in double quotes, or starting with double quotes and terminated by carriage return.

‹simple variable›

A reference to a variable excluding array variables. The reference may include a ‹type marker›

‹stream expression›

An ‹integer expression› yielding a value in the range 0..9. All character input and output is directed at streams. The output streams are :

0..7	The eight screen character streams.
8	The Printer.
9	The Cassette output file.

The input streams are :

0..7	Keyboard, any prompt sent to the corresponding screen.
8	Keyboard, any prompt sent to the Printer.
9	The Cassette input file, any prompt is ignored.

‹string expression›

An expression which yields a value of type String.

‹string variable›

A reference to a variable, which may be an array item, whose type is string.

‹type marker›

One of the characters :

%	indicating Integer
!	indicating Real
$	indicating String

‹variable›

A reference to any form of variable. This includes variables which are items in an array, in which case suitable subscripts must be included.

‹variable name›

The name of a ‹simple variable›, which may include its ‹type marker›.

ABS

Absolute value. **Function**

Use.

To determine the absolute value of a given expression.

Form.

ABS(‹numeric expression›)

Notes.

ABS of a negative value returns that value negated. ABS of a positive value returns the value unchanged.

Associated Keywords.

SGN

AFTER

Invoke subroutine after a given period has elapsed. **Command**

Use.

The system maintains a real time clock. The AFTER command allows a BASIC program to arrange for subroutines to be called at some time in the future. Four delay timers are available, each of which may have a subroutine associated with it.

Form.

AFTER ‹time delay›[,‹timer number›] GOSUB ‹line number›

The ‹time delay› is an ‹integer expression› which specifies how long before the subroutine should be called. This time is measured in 1/50ths of a second.

The ‹timer number› is an ‹integer expresssion› which specifies which of the four possible delay timers is to be used. The expression must yield a value in the range 0..3. If the expression is omitted, 0 is assumed.

Notes.

When the time specified has passed the subroutine is called automatically, just as if a GOSUB had been issued at the current position in the program. When the subroutine finishes, using a normal RETURN command, the program continues running where it left off.

The timers have different interrupt priorities. Timer 3 has the highest priority and timer 0 the lowest. See Section 9 for a discussion of the effect of priorities.

AFTER commands may be issued at any time, resetting the subroutine and time associated with the given delay timer. The delay timers are the same as those used in the EVERY command, so an AFTER overrides any previous EVERY for the given timer and vice versa.

Associated Keywords.

EVERY, REMAIN

ASC

Get ASCII value of character. **Function**

Use.

To get the numeric value of the first character of a string, given that the ASCII encoding of characters is used.

The inverse of CHR$.

Form.

ASC(‹string expression›)

Where the ‹string expression› yields a string at least one character long.

Associated Keywords.

CHR$

ATN

Arc - tangent. **Function**

Use.

To calculate the arc-tangent of a given value. The result returned is an angle either in radians (in the range −PI/2..+PI/2) or in degrees (in the range −90..+90) depending on the current mode.

Form.

`ATN(`‹numeric expression›`)`

Notes.

`DEG` and `RAD`, respectively, change BASIC to degrees or radians mode.

Associated Keywords.

`SIN, COS, TAN, DEG, RAD`

AUTO

Automatic line numbering. **Command**

Use.

To aid the input of program. Causes BASIC to generate line numbers.

Form.

AUTO [‹line number›][,‹increment›]

If the ‹line number› is omitted 10 is assumed.

The ‹increment› takes the form of a ‹line number›. If the ‹increment› is omitted 10 is assumed.

Notes.

AUTO generates the first line number given, and allows entry of program text. At end of line the next line number is automatically generated, by adding the increment to the previous number. Typing **[ESC]** terminates the AUTO action, discards the current line, and returns BASIC to Direct Mode.

If a line number is generated that is already in use, then a warning asterisk is printed after the number. The contents of the line will be replaced unless nothing is entered before the carriage return, or AUTO is terminated.

BIN$

Binary String. **Function**

Use.

To produce a string of binary digits representing the value of the given expression.

Form.

BIN$(‹unsigned integer expression›[,‹field width›])

The ‹unsigned integer expression› must yield a value in −32768..65535. This value is treated as an unsigned 16 bit value, to be converted into a string of binary digits.

The optional ‹field width› is an ‹integer expression› giving the minimum size of string to be produced. The expression must yield a value in the range 0..16.

Notes.

BIN$ always generates as many characters as are required to represent the number (zero generates at least one digit). If a ‹field width› is specified and the string is too short, then the string is filled to length with leading zeros. The string is not truncated if it is too long.

Associated Keywords.

HEX$, DEC$, STR$

BORDER

Set the Border Colour. **Command**

Use.

To change the colour of the border on the screen.

Form.

BORDER ‹colour›[,‹colour›]

If two colours are specified the border alternates between the two.

Associated Keywords.

INK

CALL

Call external subroutine. **Command**

Use.

Allows externally developed subroutines to be invoked from BASIC.

Form.

CALL ‹address expression›[,‹list of: ‹parameter›]

The ‹address expression› gives the address of the subroutine.

The ‹parameter›s may be either: ‹numeric expression› or: @‹variable›

Notes.

Parameters are passed by value.

A ‹numeric expression› parameter yielding a Real value is forced to unsigned integer.

An @‹variable› parameter passes the address of the variable.

See Appendix II for a detailed discussion of the calling sequence, variable formats etc.

Associated Keywords.

UNT

CAT

Display tape Catalogue. **Command**

Use.

The CAT command causes BASIC to start reading the cassette and to display the names of all files found.

Form.

```
CAT
```

Notes

Any open files are abandoned - any buffered output is lost.

No data is read or written, and the current program is not affected.

As files are found on tape the following is displayed :

```
FILENAME block 9 ? Ok
```

The block number given is that of the latest block read for the given file. The 'Ok' indicates that the block was read correctly. The '?' flag indicates the type of the file, and may be one of the characters :

$	a BASIC program file.
%	a Protected BASIC program file.
*	an ASCII text file.
&	a Binary file.
'	a Protected Binary file.

Other characters may appear in this field if the file was not produced by BASIC.

`CAT` may be used to verify that the files on a tape are readable, for example after a `SAVE` operation.

CHAIN
CHAIN MERGE

Chain into new program. **Command**

Use.

Allows one program to load and enter another, retaining the current variables and some or all of the current program.

Forms.

`CHAIN` ‹file name›[`,`‹line number expression›]
`CHAIN MERGE` ‹file name›[`,`[‹line number expression›]
[`,DELETE` ‹line number range›]]

Where:

The ‹file name› is a ‹string expression› giving the name of the file containing the required program.

The ‹line number expression› is a ‹numeric expression›, which when rounded to an Integer yields a value in the range −32768..65535. Negative values are then converted to their unsigned equivalent (by adding 65536), and must then be in the range 1..65535. This gives the line number to start executing from once the new program is loaded, if the line does not exist an Error 8 results. If no line number is given execution starts at the lowest numbered line.

`DELETE` causes the lines in the ‹line number range› given to be removed from the current program before the new program is loaded.

Notes.

If the file name ‹string expression› yields an empty string, then BASIC attempts to load the first file encountered on the tape.

If the first character of the file name ‹string expression› is '!', then it is removed from the filename and has the effect of suppressing the messages generated by the cassette reading software.

Both CHAIN and CHAIN MERGE have the following effects on the current environment:

- All User Functions are forgotten.
- ON ERROR GOTO is turned off.
- All open files are abandoned (any buffered output is lost).
- RESTORE action is taken.
- All active FOR, WHILE and GOSUB commands are forgotten.

During a CHAIN MERGE operation a line in the new program with the same number as an existing line will replace the existing line.

It is not possible to CHAIN MERGE a protected program from cassette.

RENUM will renumber the line numbers in the DELETE ‹line number range›, but will not affect the ‹line number expression›. In CHAIN this expression refers to a line in a separate program, so cannot be renumbered. In CHAIN MERGE BASIC cannot tell whether the ‹line number expression› refers to a line which will be unaffected by the merge, so is left consistent with CHAIN.

Associated Keywords.

LOAD, MERGE, RUN, SAVE

CHR$

Convert to character. **Function**

Use.

To convert a numeric value to its character equivalent, given that the ASCII coding of characters is used.

The inverse of ASC.

Form.

CHR$(‹integer expression›)

Where ‹integer expression› must yield a value in the range 0..255.

Associated Keywords.

ASC

CINT

Convert to integer. **Function**

Use.

To convert the given value to integer representation, if possible.

Form.

`CINT (‹numeric expression›)`

The ‹numeric expression› is rounded to integer, which must then be in the range −32768..32767.

Associated Keywords.

`CREAL, INT, FIX, ROUND, UNT`

CLEAR

Clear all variables and files. **Command**

Use.

To set all variables to zero if numeric, or null if string. Erase all arrays. Forget all User Functions. Abandon all files.

Form.

`CLEAR`

Notes.

`CLEAR` causes `BASIC` to forget any active `FOR`, `WHILE` or `GOSUB` commands.

Any buffered cassette output is lost.

CLG

Clear Graphics Screen. **Command**

Use.

To clear the graphics screen.

Form.

`CLG` [‹masked ink›]

The graphics screen is filled with the given ink. If no ink is specified then the ink specified in the latest `CLG` command is used, or ink zero is used if no `CLG` command has been executed.

Associated Keywords.

`CLS, ORIGIN`

CLOSEIN

Close the cassette input file. **Command**

Use.

To finish using cassette input file.

Form.

```
CLOSEIN
```

Associated Keywords.

OPENIN, CLOSEOUT

CLOSEOUT

Close the cassette output file. **Command**

Use.

To finish using cassette output file.

Form.

```
CLOSEOUT
```

Notes.

Any data buffered on its way to the cassette is written and the file closed.

Associated Keywords.

OPENOUT, CLOSEIN

CLS

Clear Screen Window. **Command**

Use.

To clear a given screen window to its Paper Ink.

Form.

`CLS` [`#`‹stream expression›]

The ‹stream expression› must yield a screen stream (ie a value in the range 0..7). If the ‹stream expression› is omitted stream `#0` is cleared.

Notes.

The cursor position is reset to the top left hand corner of the stream's current window (position 1,1).

Printing character `CHR$(12)` has the same effect.

CONT

Continue after **[ESC][ESC]**, STOP or END. **Command**

Use.

When a program has been halted by **[ESC][ESC]** or by a STOP or END command, the CONT command will resume program execution.

Form.

CONT

Notes

CONT will be rejected if the program has been altered in any way since it halted.

COS

Cosine. **Function**

Use.

To calculate the Cosine of a given angle. In radians mode the angle is expressed in radians. In degrees mode the angle is expressed in degrees.

Form.

`COS(`‹numeric expression›`)`

The ‹numeric expression› gives the angle in radians or degrees. The value of the argument when converted to radians, must be in the approximate range −200,000..200,000.

Notes.

With angles very much greater than 2*PI (360 degrees) the accuracy of this function becomes increasingly badly affected by the scaling of the angle into the range −PI..+PI (−180..+180 degrees). Rather than return a very inaccurate figure, BASIC will not evaluate `COS` for values much beyond the range given above.

Associated Keywords.

`SIN, TAN, ATN, DEG, RAD`

CREAL

Convert value to Real. **Function**

Use.

To convert the value of the given expression to Real representation.

Form.

`CREAL(`‹numeric expression›`)`

Associated Keywords.

`CINT`

DATA.

Declare constant data. **Command**

Form.

DATA ‹list of: ‹constant›

‹constant› may be any form of ‹numeric constant›, a ‹quoted string› or an ‹unquoted string›:

a ‹numeric constant› may be surrounded by ‹white space›, which is ignored.

a ‹quoted string› may be surrounded by ‹white space› which is ignored. The trailing double quotes may be omitted if the string is the last object on the line.

an ‹unquoted string› is between 0 and 255 characters which are taken to be a string. A comma, colon or carriage return terminates an ‹unquoted string› in this context. An ‹unquoted string› may be surrounded by ‹white space›, which is ignored.

Notes.

All the DATA commands in line number order within a program form a 'file' of constants which may be read into variables by the READ command. As each constant is read a pointer is advanced to the next one. The RESTORE command moves the pointer to a specified position in this list.

DATA commands are not executed. The constants are not checked for validity until they are READ, at which point numeric constants may only be read into numeric variables, and strings (of either type) may only be read into string variables - noting that numeric constants may be treated as ‹unquoted strings›.

Associated Keywords.

READ, RESTORE

DEC$

Generate formatted string representation of number. **Function**

Use.

To create a string representation of the value of a given expression, according to a given format template.

Form.

`DEC$(`‹numeric expression›`,`‹format template›`)`

The value of the ‹numeric expression› is to be converted to a decimal string.

The ‹format template› is a ‹string expression› which is used to control the format of the resulting decimal string.

Notes.

The value of the ‹numeric expression› is converted to a decimal string formatted according to the ‹format template›, which is interpreted as in `PRINT USING`.

The format template is not a full `PRINT USING` string, but may only contain the following characters :

`+ - $ * # , . ↑`

Associated Keywords.

`STR$, HEX$, PIN$, PRINT USING`

DEF FN

Define User Function.

Command
Illegal in Direct Mode

Use.

BASIC allows the program to define and use simple value returning functions. DEF FN is the definition part of this mechanism.

Form.

DEF FN‹function name›[(‹formal parameters›)] = ‹expression›

The ‹function name› takes the same form as a ‹variable name› (including an optional ‹type marker›), the name of the function being FN‹function name›. The value returned by the function is of the same type as the ‹function name›.

The ‹formal parameters› take the form ‹list of: ‹simple variable name›.

The ‹expression› may involve not only the formal parameters, but also other variables or functions. The result of the expression must be compatible with the type of the function.

Notes.

A user function is invoked when it is used as an argument in an expression. The invocation takes the form:

FN‹name›[(‹actual parameters›)]

The ‹actual parameters› take the form ‹list of: ‹expression›. There must be the same number of actual parameters as formal parameters. The type of each actual parameter must be compatible with the type of the corresponding formal parameter.

The formal parameters are local to the function. Variables with the same names as formal parameters are not affected by invoking the function, nor are they accessible from within the function.

If the type of the function is not explicit, then the default when the DEF FN is executed applies. If the type of a formal parameter is not explicit, then the default when the function is invoked applies.

DEFINT
DEFREAL
DEFSTR

Set default type for names. **Command**

Use.

All variable and function names have a type associated with them. This type may be explicit in the name, or implicit, depending on the defaults set by these commands.

Forms.

`DEFINT` ‹ list of:‹letter range›
`DEFREAL` ‹ list of:‹letter range›
`DEFSTR` ‹ list of:‹letter range›

where ‹letter range› is : ‹letter› or : ‹letter›—‹letter›
and the form ‹letter›—‹letter› defines an inclusive range of letters.

Notes.

When a variable name without an explicit ‹type marker› is met the current default type is assumed. The default type depends on the first character of the name. The default is set for each letter by the latest of these commands in whose ‹letter range list› the letter was included.

`DEFINT` sets the default to Integer.
`DEFREAL` sets the default to Real
`DEFSTR` sets the default to String.

The initial state of the default type for all letters is Real. The state is reset to this by:

`LOAD, RUN, CHAIN, NEW and CLEAR`

DEG

Set degrees mode. **Command**

Use.

The arithmetic functions SIN, COS, TAN and ATN work in radians by default. After a DEG command they work in degrees.

Form.

DEG

Notes.

Degrees mode persists until a RAD or CLEAR command is executed, or a new program is LOADed or RUN.

Associated Keywords.

RAD, ATN, COS, SIN, TAN

DELETE

Delete lines of program. **Command**

Use.

To remove part of the current program.

Form.

`DELETE` ‹line number range›

Notes.

Removes the lines in the given range. BASIC returns to Direct Mode after executing a `DELETE`.

Associated Keywords.

`NEW`

DI

Disable Interrupts. **Command**

Use.

Disables interrupts until re-enabled explicitly by `EI` or implicitly by end of interrupt subroutine.

Form.

```
DI
```

Notes.

Disabling interrupts ensures that no further interrupts, other than Break Interrupts, will occur until interrupts are re-enabled.

If interrupts are disabled in an interrupt subroutine they are implicitly re-enabled by the `RETURN` at the end of the subroutine.

Associated Keywords.

`EI`

DIM

Declare array dimensions. **Command**

Use.

In order to allocate memory for an array BASIC must know how many dimensions it has, and the maximum permissible value of each one. The `DIM` command allows dimensions to be set explicitly.

Form.

`DIM` ‹list of: ‹subscripted variable›

Where ‹subscripted variable› is : ‹variable name› (‹dimension list›)
and ‹dimension list› is: ‹list of: ‹integer expression›

Notes.

Each ‹integer expression› in the ‹dimension list› sets the maximum value to be allowed for the corresponding subscript. The minimum value of each subscript is 0.

Array dimensions may be set explicitly in a `DIM` command, or implicitly when the array is first encountered. When an array is declared implicitly the maximum value for each subscript is set to 10.

It is an error to attempt to change the dimensions of an array which has already been declared (explicitly or implicitly).

There is no restriction on the number of subscripts an array may have, other than line length and store size !

Associated Keywords.

`ERASE`

DRAW

Draw a line on the Graphics Screen. **Command**

Use.

To draw a line on the screen from the current position to an absolute position.

Form.

DRAW ‹x coordinate›,‹y coordinate›[,‹masked ink›]

The ‹x coordinate› and ‹y coordinate› must be ‹integer expressions›.

If no ‹masked ink› is specified the last Ink used in a DRAW, DRAWR, PLOT or PLOTR command is used. If this is the first such command then Ink 1 is used.

Notes.

This commands draws a line from the current graphics cursor position to and including the point specified. The graphics cursor is left at the destination.

While the two ends of the line may lie outside the current graphics window, only the portion inside the window is drawn.

Associated Keywords.

DRAWR, PLOT, PLOTR, MOVE, MOVER, TEST, TESTR

DRAWR

Draw a line on the Graphics Screen. **Command**

Use.

To draw a line on the screen from the current position to a position relative to it.

Form.

DRAWR ‹x offset›,‹y offset›[,‹masked ink›]

The ‹x offset› and ‹y offset› must be ‹integer expressions›.

If no ‹masked ink› is specified the last Ink used in a DRAW, DRAWR, PLOT or PLOTR command is used. If this is the first such command then Ink 1 is used.

Notes.

This command draws a line from the current graphics cursor position to and including the point whose position is given by adding the ‹x offset› to the current x coordinate, and adding the ‹y offset› to the current y coordinate. The graphics cursor is left at the destination.

While the two ends of the line may lie outside the current graphics window, only the portion inside the window is drawn.

Associated Keywords.

DRAW, PLOT, PLOTR, MOVE, MOVER, TEST, TESTR

EDIT

Edit a program line. **Command**

Use.

To amend individual program lines.

Form.

`EDIT` ‹line number›

Notes.

Enters Edit Mode with the given line as the line to be edited.
See Section 6 for a full description of Edit Mode.

EI

Enable Interrupts. **Command**

Use.

To re-enable interrupts disabled by a DI command.

Form.

EI

Notes.

Interrupts are implicitly enabled by RUN.

If interrupts are disabled in an interrupt subroutine they are implicitly re-enabled by the RETURN at the end of the subroutine.

Associated Keywords.

DI

END

End of program. **Command**

Use.

To signal the end of a program, and stop cleanly.

Form.

`END`

Notes.

Any number of `END` commands may appear anywhere in a program, though an `END` is assumed after BASIC has obeyed the last line of a program.

`END` or end of program closes all cassette files and returns to Direct Mode.

Sound generation continues until the sound queues empty.

Associated Keywords.

`STOP`

ENT

Set up Tone envelope. **Command**

Use.

While a sound is being generated it is possible to vary its tone. A tone envelope defines how the tone is to be varied.

Form.

ENT ‹envelope number›[,list of:‹envelope section›]

The ‹envelope number› is an ‹integer expression› whose absolute value must be in the range 1..15, which specifies the tone envelope to set. If the ‹envelope number› is negative, then the envelope is a repeating one.

Up to five ‹envelope section›s may be given. Each may take one of the forms :

‹step count›,‹step size›,‹pause time›
or: =‹tone period›,‹pause time›

The first form specifies an incremental change relative to the current tone period setting. The second form specifies an absolute setting for the tone period. Where :

‹step count› gives the number of steps in the section - an ‹integer expression› in the range 0..239.

‹step size› gives the amount by which to vary the tone period at each step in the envelope - an ‹integer expression› yielding a value in the range −128..+127.

‹pause time› gives the time to wait after each step - an ‹integer expression› specifying the time in 1/100th's of a second. The expression must yield a value in the range 0..255 (where 0 is treated as 256 !).

‹tone period› gives the new setting for the period - an ‹integer expression› yielding a value in the range 0..4095.

Notes.

The SOUND command sets the initial tone period, and may specify one of the fifteen tone envelopes. If an envelope which has not been set up, or no envelope is specified, then the tone remains constant throughout the sound.

When a sound with a tone envelope is started the first step of the first section is executed immediately. If, therefore, the first section is an incremental change then the tone period when the sound starts will be the initial tone period set in the SOUND command *plus* the ‹step size›. If the first section is an absolute setting then the tone period in the SOUND command is ignored.

A tone envelope has no effect on the duration of the sound. If there are steps remaining in the tone envelope when the sound finishes, they are simply abandoned. If the tone enveloepe finishes before the sound does, then the tone period retains its final shape.

A repeating tone envelope will be restarted each time it finishes until the sound terminates.

The expressions in the tone envelope are evaluated when the command is executed and the results stored away for future use. Using the tone envelope does not cause the command to be re-executed.

Each time a given tone envelope is set its previous value is lost. Changing an envelope while a sound using it is active or pending will produce indeterminate (but possibly interesting) effects.

Specifying an envelope with no sections cancels any previous setting. Any further use of the envelope will be ignored and the tone will not vary.

Associated Keywords.

ENV, SOUND

ENV

Set up Volume envelope. **Command**

Use.

While a sound is generated it is possible to vary its volume. A volume envelope defines how the volume is to be varied.

Form.

ENV ‹envelope number›[,‹list of:‹envelope section›]

The ‹envelope number› is an ‹integer expression› yielding a value in the range, 1..15, which specifies specifying the volume envelope to set.

Up to five ‹envelope section›s may be given. Each may take one of the forms :

‹step count›,‹step size›,‹pause time›
or: =‹hardware envelope›,‹envelope period›

The first form specifies an absolute or incremental section under software control, where the parameters are :

‹step count› gives the number of steps in the section - an ‹integer expression› in the range 0..127.

A step count of zero makes the section an absolute one, otherwise it is an incremental one.

‹step size› if the ‹step count› is not zero the ‹step size› gives the amount by which to vary the amplitude at each step in the envelope - an ‹integer expression› in the range −128..+127.

or : if the ‹step count› is zero then the value to set the amplitude to - that is an absolute setting.

‹pause time› gives the time to wait after each step - an ‹integer expression› specifying the time in 1/100th's of a second. The expression must yield a value in the range 0..255 (where 0 is treated as 256 !).

The second form specifies an envelope section to be executed directly by the sound hardware, where :

‹hardware envelope› is the value to be set into the envelope shape register (register 13).

‹envelope period› is the value to be set into the envelope period registers (registers 11 & 12).

Hardware envelope sections do not have an associated pause time, so the next section of the envelope is immediately executed. It is suggested that the next section has a pause of a suitable length. If there is no next section, then a pause of 2 seconds is taken.

Notes.

The SOUND command sets the initial volume, and may specify one of the fifteen volume envelopes. If an envelope which has not been set up, or no envelope is specified, then the volume remains constant throughout the sound.

When a sound using a volume envelope is started the first step of the first section of the envelope is executed immediately. If, therefore, the first section is an incremental change then the volume when the sound starts will be the initial volume set in the SOUND command *plus* the ‹step size›. If the first section is an absolute or hardware section, then the initial volume set in the SOUND command will be ignored.

Setting a step size of zero with a non-zero step count causes the current volume setting to be maintained for the duration of the section.

The expressions in the amplitude envelope are evaluated when the command is executed and the results stored away for future use. Using the amplitude envelope does not cause the command to be re-executed.

Each time a given amplitude envelope is set its previous value is lost. Changing an envelope while a sound using it is active or pending will produce indeterminate (but possibly interesting) effects.

Specifying an envelope with no sections cancels any previous setting. Any further use of the envelope will be ignored and the volume will not vary.

Associated Keywords.

ENT, SOUND

EOF

End of File test. **Function**

Use.

To test if the cassette input is positioned at end of file. Returns −1 (true) if is at end of file. Returns 0 (false) otherwise.

Form.

```
EOF
```

Notes.

If no cassette input file is open then the function returns true.

The `EOF` function may require the next block to be read from the cassette. If the reading is interrupted by pressing **ESC**, then the function returns true.

Associated Keywords.

```
OPENIN
```

ERASE

Erase arrays. **Command**

Use.

When an array is no longer required ERASE may be used to remove it and reclaim the memory it occupies ready for other use.

Form.

ERASE ‹list of: ‹variable name›

where each ‹variable name› is the name of an array, without any subscript part.

Notes.

It is an error to ERASE an array which has not been declared (explicitly or implicitly).

Once an array has been ERASEd it is possible to declare it again with new dimensions.

Associated Keywords.

DIM

ERR
ERL

Error and Error Line Numbers. **Functions**

Use.

These functions may be used in error handling subroutines to discover the error number (ERR) and the line being executed when the error occurred (ERL).

Warning.

If ERL is used in relational expressions it must be written on the left hand side for BASIC to recognise the right hand side as a line number. If the expression is written the other way round, RENUM is unable to find the line number, and will ignore it - which is likely to cause inconsistencies.

Associated Keywords.

ON ERROR, ERROR

ERROR

Cause Error action to be taken. **Command**

Use.

Invoke Error Action with a given error number. The error number may be one already used and recognised by BASIC, in which case the action taken is the same as would be taken if such an error had been detected by BASIC. Error numbers beyond those recognised by BASIC may be used by the program to signal its own errors.

Form.

`ERROR` ‹integer expression›

Where the ‹integer expression› must evaluate to a number in the range 1..255.

Notes.

In the event of BASIC attempting to print an error message for an error number that it does not recognise, the message '`Unknown error`' is produced.

Associated Keywords.

`ON ERROR, ERR, ERL`

EVERY

Invoke subroutine every time a given period has elapsed. **Command**

Use.

The system maintains a real time clock. The EVERY command allows a BASIC program to arrange for subroutines to be called at regular intervals. Four delay timers are available, each of which may have a subroutine associated with it.

Form.

EVERY ‹time period›[,‹timer number›] GOSUB ‹line number›

The ‹time period› is an ‹integer expression› which specifies how long to wait between each call of the subroutine. This time is measured in 1/50ths of a second.

The ‹timer number› is an ‹integer expresssion› specify in which of the four possible delay timers is to be used. The expression must yield a value in the range 0..3. If the expression is omitted, 0 is assumed.

Notes.

When the time specified has passed the subroutine is called automatically, just as if a GOSUB had been issued at the current position in the program. When the subroutine finishes, using a normal RETURN command, the program continues running where it left off.

The timers have different interrupt priorities. Timer 3 has the highest priority and timer Ø the lowest. See Section 9 for a discussion of the effect of priorities.

Immediately the timer expires the count is reset and the count down to the next call of the subroutine begins.

EVERY commands may be issued at any time, resetting the subroutine and time associated with the given delay timer. The delay timers are the same as those used in the AFTER command, so an EVERY overrides any previous AFTER for the given timer and vice versa.

Associated Keywords.

AFTER, REMAIN

EXP

Exponential. **Function**

Use.

Calculate e to the given power. Where e is the number whose natural logarithm is 1 (approx. 2.7182818).

Form.

EXP(‹numeric expression›)

Notes.

EXP of numbers much greater than 88 will cause an overflow error.

EXP of numbers much less than -88.7 underflows, and yields the value zero.

Associated Keywords.

LOG

FIX

Fix number to integer. **Function**

Use.

To truncate given value to an integer. Note that this does not convert floating point numbers to integer format, but simply removes any fraction part, rounding the value toward zero.

Format.

`FIX(`‹numeric expression›`)`

Associated Keywords.

`CINT, INT, ROUND`

FOR

FOR loop. **Command**

Use.

Execute a body of program a given number of times, stepping a control variable between a start and an end value.

Form.

FOR ‹simple variable›=‹start› TO ‹end› [STEP ‹step size›]

‹simple variable› is the control variable for the FOR loop, it must be an Integer or Real, and may not be an array item.

‹start› is a ‹numeric expression› giving the initial value for the control variable.

‹end› is a ‹numeric expression› giving the limiting value for the control variable.

‹step size› is a ‹numeric expression› giving the value to add to the control variable after each iteration. If the step size is omitted, a step of 1 is assumed. Note that the ‹step size› may be negative.

The results of the ‹start›, ‹end› and ‹step size› expressions are forced to the same type as the control variable.

Notes.

The FOR command initiates a FOR loop. The end of the FOR loop is marked by a NEXT command.

When the FOR command is executed the ‹start›, ‹end› and ‹step size› expressions are evaluated, then the control variable is set to the ‹start› value. If the initial value is beyond (see below) the end value then the FOR loop is skipped, otherwise the loop is entered. When the matching NEXT is found the step size is added to the control variable, and control returns to the start of the FOR loop unless the control variable is now beyond the end value.

If the step size is positive (or omitted) the FOR loop terminates when the control variable is greater than the end value. If the step size is negative the FOR loop terminates when the control variable is less than the end value.

The NEXT command which matches a given FOR is established statically when the FOR is first executed. That is to say that the NEXT which matches the FOR depends on the order of statements in the program, quite independently of the order of execution. It is not possible, therefore, to have more than one NEXT associated with a given FOR.

FOR loops may be nested.

The value of the control variable is defined once the loop has terminated. It is permissable to terminate a FOR loop by avoiding the NEXT, the value of the control variable is unchanged.

Care must be exercised if fractional step sizes are used, since the effects of rounding may mean that the control variable value can never exactly equal the final value.

Associated Keywords.

NEXT, WHILE

FRE

Free space measurement. **Function**

Use.

To establish how much free memory remains unused by BASIC. Two forms of the function exist, one causes BASIC to 'garbage collect' before measuring the free space.

Forms.

FRE(‹numeric expression›)
FRE(‹string expression›)

Notes.

The value of the argument in a FRE is not relevant, so FRE(Ø) and FRE("") are the recommended forms.

BASIC maintains a heap in the free memory, in which strings are stored. As the heap grows the free space is used up. At the same time areas within the heap area may become free, but cannot be reclaimed by BASIC until a 'garbage collection' is performed. In general BASIC does not 'garbage collect' until the heap has grown to fill all free memory.

FRE(Ø) returns the amount of memory left for the heap to grow into.

FRE("") forces BASIC to 'garbage collect' so that any unused space within the heap is returned to the free area. The result of the expression is then the maximum free space available. Note that the 'garbage collection' may take a noticeable time.

GOSUB

Go to Subroutine. **Command**

Use.

To call a given subroutine.

Form.

GOSUB ‹line number›

Notes.

When a GOSUB is executed the program jumps to the given line number, remembering the position immediately after the GOSUB so that execution may continue there when the subroutine returns.

Subroutines are terminated by a RETURN command. A subroutine may contain more than one RETURN command.

Subroutines may be nested and recurse, the depth being limited only by the size of the stack, which is big enough for subroutines nested over fifty deep. (Note, however, that other BASIC commands and functions require stack space - which will affect the nesting limit of any particular program).

Associated Keywords.

RETURN

GOTO

GOTO line. **Command**

Use.

Unconditional jump to a given line.

Form.

`GOTO` ‹line number›

HEX$

Hexadecimal string. **Function**

Use.

To produce a string of hexadecimal digits representing the value of the given expression.

Form.

HEX$(‹unsigned integer expression›[,‹field width›])

The ‹unsigned integer expression› must yield a value in −32768..65535. This value is treated as an unsigned 16 bit value, to be converted into a string of hexadecimal characters.

The optional ‹field width› is an ‹integer expression› which gives the minimum size of string to be produced. The expression must yield a value in the range 0..16.

Notes.

HEX$ always generates as many characters as are required to represent the number (zero generates at least one digit). If a ‹field width› is specified and the string is too short, then the string is filled to length with leading zeros. The string is not truncated if it is too long.

Associated Keywords.

BIN$, DEC$, STR$

HIMEM

Address of highest byte in memory used by BASIC. **Function**

Use.

The amount of memory used by BASIC may be altered by the `MEMORY` command. The built-in `HIMEM` function returns the current memory use.

Form.

```
HIMEM
```

Notes.

`HIMEM` returns a real value in the range 0...65535.

Associated Keywords.

`MEMORY`

IF

Command

Use.

To conditionally execute statements.

Forms:

IF ‹logical expression› THEN ‹option part› [ELSE ‹option part›]
IF ‹logical expression› GOTO ‹line number› [ELSE ‹option part›]

Where ‹option part› is : ‹statement(s)›
or : ‹line number›
and ‹statement(s)› is one or more statements on the same line as the IF (separated by colons).

Notes.

The ‹logical expression› is evaluated. If the result is non - zero (true) the THEN or GOTO option is executed, otherwise the THEN or GOTO option is skipped, until either end of line or a matching ELSE is found. In the latter case the ELSE option is executed.
IF statements may be nested to any depth, limited only by the line length. Each ELSE part is associated with the innermost IF which does not already have an ELSE part.

THEN ‹line number› and ELSE ‹line number› are equivalent to THEN GOTO ‹line number› and ELSE GOTO ‹line number›, respectively.

The IF command is terminated by end of line. It is not possible to have further statements independent of the IF on the same line.

Associated Keywords.

WHILE

INK

Set given Ink to the given Colour. **Command**

Use.

Depending on the current Screen Mode a number of Inks are available. The Colour, or Colours, used for an Ink may be changed by an INK command.

Form.

INK ‹ink›,‹colour›[,‹colour›]

If the second colour is specified, then the ink alternates between the two colours.

Associated Keywords.

BORDER

INKEY

Test state of key on keyboard. **Function**

Use.

INKEY may be used to test whether a given key is down, and if so, whether **[SHIFT]** and **[CTRL]** are also down.

Form.

INKEY(‹key number›)

The ‹key number› is an ‹integer expression› specifying which of the eighty keys is to be tested. The expression must yield a value in the range 0...79.

Notes

The function returns an Integer value as follows:

	Key	[SHIFT]	[CTRL]
-1	Up	Unknown	Unknown
0	Down	Up	Up
32	Down	Down	Up
128	Down	Up	Down
160	Down	Down	Down

The function tests the state of the key as established by the latest keyboard scan. The keyboard scan runs every 1/50th of a second.

The state returned is the latest known state of the keyswitch. It is independent of any key or function key definitions.

INKEY does not affect the normal generation of characters from the keyboard, or the keyboard buffering.

Associated Keywords.

INPUT, INKEY$

INKEY$

Input key from keyboard. **Function**

Use.

Input next key, if any, from keyboard. Note that any character input is not reflected to the screen.

Form.

`INKEY$`

Notes.

If there is a character pending then `INKEY$` returns it (as a one character string). If no characters are pending `INKEY$` returns an empty string.

Except for **[ESC]** (which halts the program), all characters are passed as read to the program, without reflecting them to the console.

`INKEY$` does not inspect the current state of the keyboard, but takes the next character, if any, from the keyboard interrupt buffer.

Associated Keywords.

`INPUT, INKEY`

INP

Input from I/O port. **Function**

Use.

Input value from the given I/O port.

Form.

INP(‹port number›)

The ‹port number› is an ‹address expression›.

Notes.

INP reads a byte from the given I/O port.

Associated Keywords.

OUT, WAIT

INPUT

Input data from a stream. **Command**

Use.

To read data from a given stream. Produces a prompt on the screen if input is from the keyboard.

Forms.

`INPUT` [`#`‹stream expression›`,`][`;`][‹quoted string›`;`]‹list of: ‹variable›
`INPUT` [`#`‹stream expression›`,`][`;`][‹quoted string›`,`]‹list of: ‹variable›

If the ‹stream expression› is omitted then stream `#0` (a keyboard stream) is used.

The ‹quoted string› is an optional prompt string.

Notes.

1. Keyboard Streams - Streams `#0..#8`.

The default prompt is a question mark. If an explicit prompt string is given then a question mark is appended to it if the first form is used (with a semi-colon after the prompt string), but not if the second form is used. The prompt is issued to the output stream with the same number as the input stream given.

When the `INPUT` command is executed the prompt is issued, and a line is read from the console. The line input is treated as a list of items, and BASIC attempts to assign the value of one item to each variable in the list, checking that the item is compatible with the variable. The line read takes the form:

‹list of: ‹item›

where each ‹item› may be one of:

a ‹numeric value›, which may be surrounded by ‹white space›, which is ignored. Note that a ‹numeric value› may also be treated as an ‹unquoted string›.

a ‹quoted string›, which may be surrounded by ‹white space›, which is ignored. The trailing double quotes may be omitted if the string is the last object on the line.

an ‹unquoted string›, which may be surrounded by ‹white space›, which is ignored. An ‹unquoted string› is between 0 and 255 characters terminated by comma or carriage return.

If there are the right number of items on the line, and their types are all compatible with the corresponding variable, then the values of the items are assigned to the variables.

If there are too few or too many items, or if an item is not compatible with the corresponding variable, then the message '?Redo from start' is issued, and the INPUT command restarted (so the prompt is repeated).

The optional semi-colon stops BASIC from reflecting the carriage return typed at the end of the input line. This means that the cursor is left at the end of the text just entered.

2. Cassette Stream - Stream #9.

No prompt is generated for INPUT from the cassette stream. A prompt may be specified, but it is ignored - so the same program can read from either stream type. The optional semi-colon is ignored.

BASIC attempts to read one item from the file for each variable in the list of variables. Each item must be compatible with the variable to which its value is to be assigned. Each item in the file may be one of:

a ‹numeric value›, which may be preceded by ‹white space› which is ignored. A numeric item is terminated by ‹white space›, comma, carriage return or end of file. Trailing ‹white space› is ignored. Any following comma or carriage return is ignored. Note that a ‹numeric value› may also be treated as an ‹unquoted string›.

a ‹quoted string›, which may be preceded by ‹white space›, which is ignored. All characters after the leading double quotes, including carriage returns and linefeeds, up to the trailing double quotes form the string. The string is terminated by double quotes or end of file. After the trailing double quotes trailing ‹white space› is ignored. Any following comma or carriage return is ignored.

an ‹unquoted string›, which may be preceded by ‹white space›, which is ignored. An ‹unquoted string› item is terminated by a comma, carriage return or end of file.

(Carriage return followed by linefeed is treated as a carriage return, and vice versa. Strings are limited to 255 characters, and terminate automatically after the 255th.)

Associated Keywords.

LINE INPUT, READ, INKEY$

INSTR

Search for substring in string. **Function**

Use.

To search from a given point in a given string for the first occurrence of another given string.

Form.

`INSTR([`‹start position›`,]`‹searched string›`,` ‹searched for string›`)`

The ‹start position› is an ‹integer expression› giving the character position within the searched string at which to start searching. The expression must yield a value in the range 1..255. If omitted then the search starts at the first character of the searched string.

The ‹searched string› is a ‹string expression› giving the string to be searched along.

The ‹searched for string› is a ‹string expression› giving the string to search for.

Notes.

If the ‹searched for string› is found, `INSTR` returns the position within the ‹searched string› of the start of the first occurrence of the ‹searched for string›. Otherwise `INSTR` returns zero.

If there are no characters in the ‹searched string› from the given (or default) ‹starting position›, then `INSTR` always returns zero.

If the ‹searched for string› is null, then it is immediately found, unless the previous rule applies.

INT

Number to integer. **Function**

Use.

To round a given value to the nearest smaller integer (round toward minus infinity). Note that this does not convert Real numbers to Integer format, but simply removes any fraction part.

(For positive numbers this is equivalent to FIX. For negative numbers the value returned is one less than FIX would return, unless the value was already integral.)

Form.

INT(‹numeric expression›)

Associated Keywords.

CINT, FIX, ROUND

JOY

Inspect Joystick status. **Function**

Use.

Two Joysticks are supported. This function directly tests the current state of a given Joystick.

Form.

JOY(‹joystick number›)

The ‹joystick number› is an ‹integer expression› which must yield a value in the range 0..1, specifying which of the two joysticks to test.

Notes.

The JOY function returns an Integer value which is 'bit significant', each bit reflecting the current state of one of the switches, as follows :

Bit 0: **Up**
1: **Down**
2: **Left**
3: **Right**
4: **Fire 2**
5: **Fire 1**

Both joysticks are essentially treated as part of the keyboard. Joystick 0 is separate from the keyboard, while joystick 1 is mapped onto an area on the main keyboard. The JOY function is similar to the INKEY function, except that the state of several 'keys' is returned at once.

The function tests the state of the joystick as established by the latest keyboard scan. The keyboard scan runs every 1/50th of a second.

The state returned is the latest known state of the joystick. It is independent of any key or function key definitions.

JOY does not affect the normal generation of characters from the keyboard.

Associated Keywords.

INKEY

KEY

Set new function key definition. **Command**

Use.

Thirty two keyboard 'expansion tokens' are supported. When one of these is read it is expanded to the string of characters associated with it. The KEY command associates a string with a given expansion token.

Form.

KEY ‹expansion token number› , ‹string expression›

The ‹expansion token number› is an ‹integer expression› which must yield a value in the range 0..31, specifying which of the expansion tokens is to be set.

The ‹string expression› is evaluated and the result stored for future expansion of the token.

Notes.

The thirty two expansion tokens are the characters with values in the range 128..159. When the machine is first turned on the numeric keypad is set to the characters 128..139 which are set to expand to digits, dot and carriage return. **[CTRL] - [ENTER]** is set to character 140, set to expand to RUN"‹carriage return›. The other expansion tokens are unused, and have empty strings associated with them.
Provision is made for a total of 120 characters in all the strings associated with the expansion tokens.

Using the KEY DEF command other keys may be turned into function keys.

Associated Keywords.

KEY DEF

KEY DEF

Define Key value. **Command**

Use.

The value generated when a key is pressed may be redefined using this command. Whether a key repeats or not may also be set.

Form.

KEY DEF ‹key number›,‹repeat›[,‹normal›[,‹shifted›[,‹control›]]]

The ‹key number› is an ‹integer expression› yielding a value in the range 0..79, specifying which key to redefine.

The ‹repeat› parameter is an ‹integer expression› yielding a value in the range 0..1, specifying whether the key is to repeat: 1 enables repeat, 0 disables it.

The ‹normal›, ‹shifted› and ‹control› parameters, if present, are all ‹integer expression›s, yielding values in the range 0..255. These parameters redefine the value to be generated when the key is pressed :

‹normal› the value when neither **[SHIFT]** nor **[CTRL]** is also pressed

‹shifted› the value when **[SHIFT]**, but not **[CTRL]**, is also pressed

‹control› the value when **[CTRL]** is pressed

Where no new value is specified the previous value is retained.

Notes.

The values generated by keys are interpreted as follows :

0..31	These values tend to have special effects when printed, but are treated as ordinary characters coming from the keyboard except for &0D (13) which is Carriage Return.
32..127	Ordinary characters, usually interpretted as ASCII
128..159	The expansion tokens which are expanded according to the strings set by KEY commands.
160..223	Ordinary characters.
224..254	Special values used for **[ESC]**, edit and copy cursor keys.
255	Ignored

It is inadvisable to redefine the carriage return, **[ESC]** and cursor keys, since this adversly affects the behavior of BASIC.

Redefining the **[SHIFT]** and **[CTRL]** keys is not effective.

Associated Keywords.

KEY

LEFT$

Extract left hand part of string. **Function**

Use.

To extract a given number of characters from the left hand end of a given string.

Form.

LEFT$(‹string expression›,‹required length›)

The ‹string expression› gives the string from which to extract characters.
The ‹required length› is an ‹integer expression› giving how many characters to extract. The expression must yield a value in the range 0..255.

Notes.

If the given string is shorter than the ‹required length›, then the entire string is returned. Otherwise the left hand end of the string is returned, giving a string of the ‹required length›.

Associated Keywords.

MID$, RIGHT$

LEN

Determine length of string. **Function**

Use.

To determine the length of a given string.

Form.

LEN (string expression)

Notes.

LEN returns an integer in the range 0..255. Ø indicates that the string is null.
All characters in the string are counted, including any non-printing ones.

LET

Preface an assignment. **Command**

Use.

The LET command is not very useful, it is really a hangover from the earliest BASICs, and may be ignored.

Form.

LET ‹variable›=‹expression›

Notes.

The expression is evaluated and the resulting value assigned to the variable, unless the two are incompatible.

The LET keyword is, in fact, wholly redundant.

LINE INPUT

Input complete line from a stream. **Command**

Use.

To read a complete line of text from the given stream to a given string variable. Produces a prompt on the screen if the input is from the keyboard.

Forms.

`LINE INPUT` [#‹stream expression›,][;][‹quoted string›;]‹string variable›
`LINE INPUT` [#‹stream expression›,][;][‹quoted string›,]‹string variable›

If the ‹stream expression› is omitted input is from keyboard stream #0.

The ‹quoted string› is an optional prompt string.

The ‹string variable› gives the name of the variable to which to assign the string created.

Notes.

1. Keyboard Streams - Streams #0...#8.

The default prompt is a question mark. If an explicit prompt string is given then a question mark is appended to it if the first form is used (with a semi-colon after the prompt string), but not if the second form is used. The prompt is issued to the output stream with the same number as the keyboard input stream.

When the `LINE INPUT` command is executed the prompt is issued, and a line is read from the keyboard and assigned to the given string. The line is terminated by carriage return (or after 255 characters, whichever is the sooner).

If the optional semi-colon is present then BASIC will not reflect the carriage return typed at the end of the input line. This means that the cursor is left at the end of the text just entered.

2. Cassette Input Stream - Stream #9.

No prompt is generated for `LINE INPUT` from the cassette input stream. A prompt may be specified, but it is ignored - so the same program can read from either stream type. The optional semi-colon is ignored.

When the `LINE INPUT` command is executed a line is read from the stream and assigned to the given string. The line is terminated by carriage return (or after 255 characters, whichever is the sooner).

Associated Keywords.

`READ, INPUT, INKEY$, INPUT$`

LIST

List program. **Command**

Use.

To list the current program to the given stream. Part or all of the program may be listed.

Form.

LIST [‹line number range›][,#‹stream expression›]

The ‹line number range› defines which lines of the program to list. If it is omitted then all the program is listed.

The ‹stream expression› defines where to send the listing. If it is omitted the listing is sent to stream #0.

Notes.

BASIC returns to Direct Mode after obeying a LIST command.

Pressing **[ESC]** will suspend the listing, which may then be resumed by pressing **[SPACE]**. **[ESC][ESC]** will cause BASIC to abandon the listing and return immediately to Direct Mode.

LOAD

Load a program into memory. **Command**

Use.

To read a BASIC program from cassette into memory, replacing any existing program, or to load a binary file into memory.

Form.

`LOAD` ‹file name›[`,`‹address expression›]

The ‹file name› is a ‹string expression› giving the name of the file from which to load.

If the file is a binary file, then it will be loaded into memory at the address from which it was written, unless the second parameter is given, specifying the address to load at.

Notes.

If the ‹string expression› yields an empty string, then BASIC attempts to load the first file encountered on the tape.

If the first character of the ‹string expression› is '!', then it is removed from the filename and has the effect of suppressing the messages generated by the cassette reading software.

If a BASIC program is loaded then any existing program, user functions and variables are deleted from memory. `DEFINT`, `DEFREAL` and `DEFSTR` settings are reset.

Binary files may only be loaded into memory outside BASIC's current memory pool - above `HIMEM` for instance.

Any cassette files are abandoned - any buffered output is lost.

Associated Keywords

`SAVE, RUN, CHAIN, CHAIN MERGE, MERGE`

LOCATE

Set a new cursor location. **Command**

Use.

LOCATE moves the cursor to a new position.

Form.

LOCATE [#‹stream expression›,]‹x coordinate›,‹y coordinate›

If present the ‹stream expression› must yield a screen stream. If omitted stream #0 is assumed.

The ‹x coordinate› and ‹y coordinate› are ‹integer expression›s which must yield values in the range 1..255. The ‹x coordinate› gives the new character position required, the ‹y coordinate› the new line number.

Notes.

The new position is relative to the current window origin. The top left hand corner of a window is location 1,1.

If the cursor is moved outside the current window it remains there until it is turned on or a character printed, whereupon it is forced into the window. The cursor is off unless BASIC is waiting for input from the keyboard, when it is on unless it is disabled by the user. Appendix IV contains a fuller description of cursor positioning.

Associated Keywords.

WINDOW

LOG

Natural logarithm. **Function**

Use.

To calculate natural logarithms.

Form.

LOG (‹numeric expression›)

where the result of the ‹numeric expression› must be greater than zero.

Associated Keywords.

EXP, LOG10

LOG10

Logarithm, base 10. **Function**

Use.

To calculate logarithms to base 10.

Form.

LOG10(‹numeric expression›)

where the result of the ‹numeric expression› must be greater than zero.

Associated Keywords.

EXP, LOG

LOWER$

Convert string to lower case. **Function**

Use.

To create a new string which is a copy of another, with all upper case alphabetic characters converted to their lower case equivalents.

Form.

`LOWER$(`‹string expression›`)`

Notes.

The result of the ‹string expression› is returned, with any characters in the range 'A'..'Z' converted to the equivalent character in the range 'a'..'z'.

Associated Keywords.

`UPPER$`

MAX

Determine maximum value. **Function**

Use.

To determine the maximum of a number of values.

Form.

`MAX` (‹list of: ‹numeric expression›)

Notes.

`MAX` returns the value of the largest of the ‹numeric expression›s.

Associated Keywords.

`MIN`

MEMORY

Reset BASIC memory parameters. **Command**

Use.

To change the amount of memory used by BASIC.

Form.

`MEMORY` ‹address expression›

The ‹address expression› gives the address of the highest byte in memory which may be used by BASIC.

Notes.

The `HIMEM` function returns the current highest byte in memory.

The `MEMORY` command interacts with `SYMBOL AFTER` and Cassette files, since these automatically allocate space for themselves at the top of memory (see Appendix IX for a fuller description of the interaction).

Associated Keywords.

`HIMEM`

MERGE

Merge program from cassette into current program. **Command**

Use.

To add the contents of a file to the current program in memory.

Form.

`MERGE` ‹file name›

The ‹file name› is a ‹string expression› giving the name of the file to be read.

Notes.

If the filename ‹string expression› yields an empty string, then BASIC attempts to merge the first file encountered on the tape.

If the first character of the filename ‹string expression› is '!', then it is removed from the filename and has the effect of suppressing the messages generated by the cassette reading software.

`MERGE` has the following effects on the current environment :

- All variables and arrays are discarded.
- All User Functions are forgotten.
- `ON ERROR GOTO` is turned off.
- All open files are abandoned - any buffered output is lost.
- `RESTORE` action is taken.
- `DEFINT, DEFREAL & DEFSTR` settings are reset.

During a `MERGE` operation a line in the new program with the same number as an existing line will replace the existing line.

BASIC returns to Direct Mode after executing a `MERGE` command.

It is not possible to merge a protected program from cassette.

Associated Keywords.

`LOAD, CHAIN, CHAIN MERGE`

MID$

Replace part of a string. **Command**
Return a part of a string. **Function**

Use.

MID$ specifies part of a string (a sub-string) which can be used either as the destination of an assignment (MID$ as a Command) or as an argument in a string expression (MID$ as a Function).

Form.

MID$(‹string›,‹start position›[,‹sub-string length›])

For MID$ as a Command ‹string› must be a ‹string variable›, part of which is to be altered.

For MID$ as a Function ‹string› is a ‹string expression›, part of which will be returned as the function's value.

The ‹start position› is an ‹integer expression› specifying the position of the character in ‹string› which is to be the first character of the sub-string. The expression must yield a value in the range 1..255.

The ‹sub-string length› is an ‹integer expression› specifying the length of the sub-string. If omitted the sub-string extends to the end of the original string. The expression must yield a value in the range 1..255.

Notes.

The sub-string specified in MID$ is defined by a starting character position and a length. The first character of the original string is at position 1. The length defaults to all characters after the starting position of the sub-string. If the starting position specified is beyond the end of the string, the sub-string is null. Regardless of the length specified the sub-string always ends at the end of the original string.

When MID$ is used as a Command it appears on the left hand side of an assignment. On the right hand side must be a ‹string expression›. The sub-string specified by the MID$ is replaced by the value of the ‹string expression›. If the ‹string expression› yields a string that is shorter than the sub-string, then the extra characters in the sub-string are unaffected. If the ‹string expression› yields a string that is longer than the sub-string, then the excess characters are discarded. The sub-string to be assigned to may not be zero length.

Associated Keyword.

LEFT$, RIGHT$

MIN

Determine minimum value. **Function**

Use.

To determine the minimum of a number of values.

Form.

`MIN` (‹list of: ‹numeric expression›)

Notes.

`MIN` returns the value of the smallest of the ‹numeric expression›s.

Associated Keywords.

`MAX`

MODE

Set screen mode. **Command**

Use.

To change the mode of the screen.

Form.

MODE ‹integer expression›

The expression must yield a value in the range 0..2.

Notes.

The screen modes are as follows :

0 :	16 Colour Mode.	25 lines by 20 characters,	200 dots by 160 dots.
1 :	4 Colour Mode.	25 lines by 40 characters,	200 dots by 320 dots.
2 :	2 Colour Mode.	25 lines by 80 characters,	200 dots by 640 dots.

The screen is cleared to Ink 0 (which is not necessarily the current Paper Ink). All windows are reset to the whole screen, all cursors are reset to the top left corner (position 1,1). The graphics window is set to the whole screen and the graphics origin and graphics cursor are set to the bottom left corner (position 0,0).

Associated Keywords.

WINDOW, ORIGIN

MOVE

Move Graphics Cursor - Absolute. **Command**

Use.

To move the Graphics Cursor to a position specified by absolute coordinates.

Form.

MOVE ‹x coordinate› , ‹y coordinate›

The ‹x coordinate› and ‹y coordinate› must be ‹integer expressions›.

Associated Keywords.

MOVER, PLOT, PLOTR, DRAW, DRAWR, TEST, TESTR

MOVER

Move Graphics Cursor - Relative. **Command**

Use.

To move the Graphics Cursor to a position specified relative to the current position.

Form.

MOVER ‹x offset› , ‹y offset›

The ‹x offset› and ‹y offset› must be ‹integer expressions›.

Notes.

The Graphics Cursor is moved to the position whose coordinates are given by adding the ‹x offset› to the current x coordinate, and the ‹y offset› to the current y coordinate.

Associated Keywords.

MOVE, PLOT, PLOTR, DRAW, DRAWR, TEST, TESTR

NEW

Prepare for a new program. **Command**

Use.

Before starting a new program `NEW` may be used to completely clear the current contents of memory.

Form.

`NEW`

Notes.

As well as deleting the program in memory, the following are affected:

- All variables and arrays are discarded
- All User Functions are forgotten.
- `ON ERROR GOTO` is turned off.
- All files are abandoned - any buffered output is lost.
- `DEFINT, DEFREAL & DEFSTR` settings are reset.
- `TRON` is turned off.

BASIC always returns to Direct Mode after a `NEW` is executed.

NEXT

Step FOR control variable to next value. **Command**

Use.

Delimits the end of a FOR loop. The NEXT command may be anonymous, or may refer to its matching FOR.

Form.

NEXT [‹list of: ‹variable›]

where: NEXT ‹variable› , ‹list of: ‹variable›
is equivalent to: NEXT ‹variable› : NEXT ‹list of: ‹variable›

Notes.

The NEXT command defines the end of a FOR loop. The way in which FOR and NEXT are tied together is described under FOR. When a NEXT command is encountered BASIC knows which (if any) FOR it must be associated with. If the NEXT is followed by a variable name, then that name be the same as the name of the control variable in the matching FOR command.

Associated Keywords.

FOR

ON ‹expression› GOSUB
ON ‹expression› GOTO

Computed GOSUB & GOTO. **Command**

Use.

To choose one of a number of subroutines to call or a number of lines to jump to, depending on the result on an expression.

Forms.

ON ‹selector› GOSUB ‹list of: ‹line number›
ON ‹selector› GOTO ‹list of: ‹line number›

The ‹selector› is an ‹integer expression› which must yield a value in 0..255.

Notes.

The value of the ‹selector› selects a line number from the list. The value 1 selects the 1st, 2 the 2nd, and so on. The subroutine starting at the selected line number is called, or the selected ‹line number› is jumped to. Zero, or any result greater than the number of ‹line numbers› in the list, will do nothing.

Null line number entries in the list are legal. Selecting a null entry is not (Error 2).

Associated Keywords.

GOTO, GOSUB

ON BREAK GOSUB

Enable Break Interrupt Subroutine. **Command**

Use.

When the **[ESC]/[ESC]** sequence is pressed while BASIC is running a program execution is suspended, and BASIC returns to Direct Mode. The ON BREAK GOSUB enables an interrupt subroutine which will be invoked instead.

Form.

ON BREAK GOSUB ‹line number›

Notes.

The ON BREAK interrupt has the highest priority of all interrupts. See Section 9 for a description of the priority structure.

The RETURN command may be used to terminate the subroutine in the usual way.

If the interrupt subroutine wishes to suspend program execution the STOP command may be used.

The interrupt subroutine remains enabled until an ON BREAK STOP command is executed.

Each ON BREAK GOSUB command overrides the previous ON BREAK GOSUB or ON BREAK STOP setting, and takes effect on the next **[ESC]/[ESC]** interrupt.

Associated Keywords.

ON BREAK STOP

ON BREAK STOP

Disable Break Interrupt Subroutine. **Command**

Use.

Turn off any **[ESC]/[ESC]** trap set by an `ON BREAK GOSUB` command.

Form.

```
ON BREAK STOP
```

Notes.

An `ON BREAK STOP` command may be issued in an `ON BREAK` interrupt subroutine. This will disable the trap, but has no other immediate effect. `STOP` must be used if the subroutine wishes to stop the program.

Associated Keywords.

`ON BREAK GOSUB`

ON ERROR GOTO

Set Error Trap. **Command**

Use.

When BASIC detects an error while obeying a command it can either take the default action, which is to generate an error message and return to Direct Mode, or it can invoke an error handling subroutine in the program. `ON ERROR GOTO` sets BASIC into one of these states.

Forms.

`ON ERROR GOTO` ‹line number›
`ON ERROR GOTO Ø`

The ‹line number› specifies the line to which control is to be transferred in the event of an error.

Notes.

`ON ERROR GOTO` ‹line number› enables error trapping. `ON ERROR GOTO Ø` disables error trapping - note the special effect of this during error processing, see below.

If an error occurs in Program Mode while error trapping is enabled control is transfered to the ‹line number› specified in the `ON ERROR GOTO` command. BASIC is then in Error Processing Mode. The functions `ERR` and `ERL` indicate which error has occurred, and which line it occurred in.

`ON ERROR GOTO Ø` in Error Processing Mode not only disables error trapping, but also causes BASIC to take the default error action for the error that was trapped. Error processing routines may, therefore, cope with those errors for which they are competent and invoke the default action for the rest.

If an error is detected during Error Processing Mode BASIC immediately takes the default action.

The `RESUME` command is only legal during Error Processing Mode, and allows the program to resume execution in one of three ways :

- at the statement in which the error occured
- at the statement after the one in which the error occured
- at a specified line number

Associated Keywords.

`ERR, ERL, RESUME`

ON SQ GOSUB

Set sound queue interrupt. **Command**

Use.

Enable an interrupt for when there is a free slot in the given sound queue.

Form.

`ON SQ(`‹channel›`) GOSUB` ‹line number›

The ‹channel› is an ‹integer expression› yielding one of the values :

1 : for channel A
2 : for channel B
4 : for channel C

Notes.

Each of the three channels has a separate interrupt, though they all have the same priority - see Section 9 for a discussion of priorities.

The `ON SQ` interrupt subroutine is called when the queue for the given channel is not full - the interrupt may occur immediately it is enabled if the queue is not then full. The interrupt is disabled when it occurs, so the subroutine must issue an `ON SQ` itself if the channel is to continue producing interrupts.

The `SOUND` command and `SQ` function both disable `ON SQ` interrupts.

An `ON SQ` subroutine ends in a `RETURN` in the usual way.

Associated Keywords.

`SOUND, SQ`

OPENIN

Open cassette file for input. **Command**

Use.

To open a cassette file for input.

Form.

`OPENIN` ‹file name›

‹file name› is a ‹string expression› giving the name of the file to be opened, the first sixteen characters of which are significant.

Notes.

If the first character of the filename ‹string expression› is '!', then it is removed from the filename and has the effect of suppressing the messages generated by the cassette reading software.

When the `OPENIN` command is obeyed the cassette is searched for the required file, and the first block read from tape ready for processing.

Associated Keywords.

`CLOSEIN, OPENOUT`

OPENOUT

Open cassette file for output. **Command**

Use.

To open a cassette file for output.

Form.

OPENOUT ‹filename›

‹file name› is a ‹string expression› giving the name of the file to be written, the first sixteen characters of which are significant.

Notes.

If the first character of the filename ‹string expression› is '!', then it is removed from the filename and has the effect of suppressing the messages generated by the cassette writing software.

Cassette output creates a new file on tape, with the name given. Cassette files are organised in blocks of 2K bytes each. Nothing is written to tape until a complete block is ready and another character is written to the buffer, or until the file is closed.

Associated Keywords.

CLOSEOUT, OPENIN

ORIGIN

Set graphics screen origin and window. **Command**

Use.

The position 0,0 in the graphics coordinate system may be moved. The area of the screen used by the graphics may be restricted.

Form.

`ORIGIN` ‹x› , ‹y›[, ‹left› , ‹right› , ‹top› , ‹bottom›]

The ‹x› and ‹y› coordinates of the new origin must be ‹integer expressions›.

The ‹left›,‹right›,‹top› and ‹bottom› positions specify the graphics window. If they are omitted the current window remains unaffected.

Notes.

The new origin coordinates are absolute screen positions, where the bottom left of the screen is 0,0 and coordinates grow up and right.

Each window parameter specifies the last position in the given direction to be included in the window. These positions are given as absolute screen positions, where the bottom left of the screen is 0,0 and coordinates grow up and right.

The order of the ‹left› and ‹right› parameters may be reversed (since ‹right› must be greater than, or equal to, ‹left›).

The order of the ‹top› and ‹bottom› parameters may be reversed (since ‹top› must be greater than, or equal to, ‹bottom›).

If any of the window parameters specify positions off the screen, then they are taken as the extreme position in the given direction.

The ‹left› parameter is truncated to a multiple of eight. The ‹right› parameter is forced up to the nearest multiple of eight minus one. (This forces the graphics window to be a whole number of bytes across, which allows for various optimisations in the screen handling software.)

The ‹bottom› parameter is truncated to a multiple of two. The ‹top› parameter is forced up to the nearest multiple of two minus one. (This forces the graphics window to be a whole number of pixels high.)

Associated Keywords.

`WINDOW`

OUT

Output a value to a processor output port. **Command**

Use.

To send a value to a given I/O port.

Form.

OUT ‹port number› , ‹integer expression›

The ‹port number› is an ‹address expression›.

The ‹integer expression› gives the data byte to be sent to the given output port. The expression must yield a value in the range 0..255.

Associated Keywords.

INP, WAIT

PAPER

Set background ink for characters. **Command**

Use.

When characters are put to the screen the character cell is filled with the Paper ink before the character is written - unless Transparent mode has been selected. The PAPER command specifies the ink to be used.

Form.

PAPER [#‹stream expression›,]‹masked ink›

If present, the ‹stream expression› must yield a screen stream. If the ‹stream expression› is omitted, stream #0 is assumed.

Notes.

Only Mode 0 supports sixteen inks. In the other modes the value of the ‹masked ink› yields a smaller, suitable, ink number.

Associated Keywords.

PEN

PEEK

Peek at memory location. **Function**

Use.

To read a given byte of the machine's memory.

Form.

PEEK(‹address expression›)

Notes.

PEEK returns a value in the range 0...255.

PEEK reads from the machine's read/write memory, so can read from the screen memory under the 'upper ROM' and the memory under the 'lower ROM'.

Associated Keywords.

POKE

PEN

Set the foreground ink for characters. **Command**

Use.

To set the ink to be used when drawing characters on the screen.

Form.

PEN [#‹stream expression›,]‹masked ink›

If present, the ‹stream expression› must yield a screen stream. If the ‹stream expression› is omitted, stream #0 is assumed.

Notes.

Only Mode 0 supports sixteen inks. In the other modes the value of the ‹masked ink› yields a smaller, suitable, ink number.

Associated Keywords.

PAPER

PI

Function

Use.

Returns the closest possible machine representation to the value of pi.

Form.

```
PI
```

Notes.

Pi is approximately `3.1415926535898`. The closest machine representation is approximately `3.1415926534683`.

PLOT

Plot a point on the Graphics Screen. **Command**

Use.

To plot a point on the screen at an absolute position.

Form.

PLOT ‹x coordinate› , ‹y coordinate›[, ‹masked ink›]

The ‹x coordinate› and ‹y coordinate› must be ‹integer expression›s.

If no ink is specified the last ink used in a DRAW, DRAWR, PLOT or PLOTR command is used. If this is the first such command then ink 1 is used.

Notes.

A point is plotted at the absolute position specified. The graphics cursor is left at that position.

Associated Keywords.

DRAW, DRAWR, PLOTR, MOVE, MOVER, TEST, TESTR

PLOTR

Plot a point the Graphics Screen. **Command**

Use.

To plot a point at a position specified relative to the current graphics cursor position.

Form.

PLOTR ‹x offset› , ‹y offset›[, ‹masked ink›]

The ‹x offset› and ‹y offset› must be ‹integer expression›s.

If no ink is specified the last ink used in a DRAW, DRAWR, PLOT or PLOTR command is used. If this is the first such command then ink 1 is used.

Notes.

A point is plotted at the position given by adding the ‹x offset› to the current x coordinate, and adding the ‹y offset› to the current y coordinate. The graphics cursor is left at the target position.

Associated Keywords.

DRAW, DRAWR, PLOT, MOVE, MOVER, TEST, TESTR

POKE

Poke a value into machine memory. **Command**

Use.

Allows direct write access to machine memory.

Form.

POKE ‹address expression› , ‹integer expression›

The ‹address expression› gives the address of the byte to be written to.

The ‹integer expression› gives the value to be written to the byte at the given address. The expression must yield a value in the range 0..255.

Associated Keywords.

PEEK

POS

Stream position. **Function**

Use.

To establish the current position on a given stream.

Format.

POS(#‹stream expression›)

Notes.

1. Screen Streams - Streams #0–#7

POS returns the current 'x' coordinate of the cursor. Note that this is relative to the current window origin. The top left hand corner of a window is location 1,1.

2. Printer Stream - Stream #8

Returns the current position across the printer. The left hand edge of the printer is position 1. BASIC counts all characters whose values are greater than &1F (31) into the position.

3. Cassette Output Stream - Stream #9

A logical position on file streams is maintained by counting all printing characters sent to the stream since the last carriage return. Printing characters are characters whose values are greater than &1F (31). The position immediately after a carriage return is position 1.

Associated Keywords.

VPOS

PRINT

Print to a stream. **Command**

Use.

To print data, numeric or string, to a given stream.

Form.

`PRINT` [#‹stream expression›`,`][‹print list›][‹using clause›][‹separator›]

The ‹stream expression› gives the stream on which to print. If it is omitted stream `#0` is assumed.

‹print list› is: ‹print item›[‹separator›‹print item›]*
where ‹print item› is: ‹expression›
or: `SPC`(‹integer expression›)
or: `TAB`(‹integer expression›)

‹using clause› is: `USING` ‹string expression›`;` ‹using list›
where ‹using list› is: ‹expression›[‹separator›‹expression›]*

‹separator› is: comma or semi-colon.

(Note that the [..]* metalanguage construct means that the object is optional, but may repeated any number of times.)

1. Effect.

The ‹print items›, if any, in the ‹print list› are evaluated and printed in 'free format'.

If a ‹using clause› is present, then the ‹string expression› defines a template which controls the format in which the values of the expressions in the ‹using list› are printed.

When all the arguments have been processed a carriage return is issued, unless the `PRINT` command ends in a ‹separator›, `TAB` or `SPC`.

2. Free Format Printing.

The ‹print items› are evaluated in turn and their results printed. Numeric results are printed as described below. String results are sent to the stream character by character. `SPC` and `TAB` are described below.

A comma following a ‹print item› causes BASIC to space forward to the start of the next print zone before processing the next expresssion. A semi-colon simply separates expressions.

2.1 *Print Zones and Stream Width.*

BASIC divides the console line into zones, whose width is set by the **ZONE** command, and which by default are 13 characters wide. When **PRINT** spaces forward to the next print zone BASIC will start a new line if there are less than 'zone width' characters to the edge of the stream.

Before printing the result of an expression BASIC checks that there is enough room for it between the current position and the edge of the stream, unless the current position is the beginning of the line. If there is not enough room, a new line is started before the result is printed.

The 'edge of a stream' depends on the stream type :

Screen Streams :	The edge is the edge of the current window.
Printer Stream :	The edge is defined by the **WIDTH** command. The position on the printer is as defined in **POS**.
Cassette Stream :	No edge exists.

2.2 *Free Format Number Printing.*

Positive numbers are preceded by a space; negative numbers by a minus sign. All numbers are followed by a space.

All numbers are printed with the minimum of characters. No decimal point is printed if there are no significant fraction digits. At least one digit is printed before any decimal point - so numbers may include a leading zero.

Real numbers are printed in scaled format (with an exponent) if they cannot be accurately represented by a number with nine or less digits (excluding a possible leading zero). The scaled format used gives only six significant digits, if greater precision is required then a **USING** Format must be used.

2.3 **SPC** *Print Function.*

Use.

SPC prints a given number of spaces.

Form.

SPC (‹integer expression›**)**

If ‹integer expression› yields a negative value, zero is assumed. If the value is greater than the stream width, then it is reduced to the range 1..stream width by a suitable number of subtractions of the stream width. The resulting value gives the number of spaces to print.

Notes.

The given number of spaces are printed, starting unconditionally at the current position. **SPACE$** is not quite equivalent, since BASIC would check that the string fitted on the current line, and possibly issue a carriage return before the spaces.

SPC need not be followed by a comma or semi-colon, a semi-colon is assumed (including when SPC is the last item in the print command).

2.4 TAB Print Function.

Use.

TAB prints spaces to move to the the given print position.

Form.

TAB(‹integer expression›)

If ‹integer expression› yields a value less than one, one is assumed. If the value is greater than the stream width, then it is reduced to the range 1..stream width by a suitable number of subtractions of the width. The result gives the print position to which to move.

Notes.

If the required print position is greater than or equal to the current position, spaces are printed until the the required position is reached (this may print nothing at all).

If the required print position is less than the current position, BASIC issues a carriage return followed by spaces to reach the required position on the new line.

TAB need not be followed by a comma or semi-colon, a semi-colon is assumed (including when TAB is the last item in the print command).

3. USING Formatted Printing.

The expressions in the ‹using list› are evaluated in turn, and printed according to the format template. After each expression has been evaluated the template is processed to find a suitable format specification for its result. If the template string is exhausted while searching for a format specification the search is restarted from the beginning of the string. If no suitable format is found an error (Error 5) is generated.

Note that unlike Free Format printing the USING Formatted Printing does not perform any print zoning or edge checking.

3.1 Format Template.

The format template is a string which is interpreted character by character to control the way in which the result of each expression is printed. The meaning of characters in the template is described below. The following characters are recognised in format specifications :

! \ & # . + − * $ ↑ ,_

Other characters will be printed if they are encountered while searching the template for a format specification. If one of the above characters is encountered out of context then it too will be printed. The '_' character is not printed, but causes the following character to be printed 'as is'.

3.2 Format Specifications suitable for Strings.

'!' Only the first character of the string is printed.

'\‹spaces›\'
The first 'n' characters of the string are printed, where the string '\‹spaces›\' is 'n' characters long.

'&' The complete string is printed 'as is'.

3.3 Format Specifications suitable for Numbers.

Numbers may be printed without an exponent part (unscaled) or with one (scaled). The template for a number may not exceed 20 characters, not counting the exponent and trailing sign options.

Numbers are rounded to the number of digits printed.

The body of the number:

'#' Each '#' specifies a digit position.

'.' Specifies the position of the decimal point. There may be at most one '.' in a number format specification.

',' May appear before '.'. Specifies a digit position, and also requests that digits before the decimal point be divided into groups of three, separated by commas.

Leading Dollar and Asterisk options:

The following are mutually exclusive options. These options must be specified immediately before the body of the number :

'$$' Specifies two digit positions. Specifies that a '$' sign be printed immediately before the first digit or decimal point (after any leading sign). Note that the '$' will occupy one of the 'digit' positions.

'**' Specifies two digit positions. Specifies that any leading spaces be replaced by '*'s.

'**$' Specifies three digit positions. Acts as the previous two options combined.

Sign Options:

The default is for '-' to be printed immediately before the number (and any leading dollar) if the number is negative, and no sign at all if the number is positive. The '-' will occupy one of the digit positions before the decimal point.

It is possible to specify that '+' be printed for positive numbers, and that
the sign is to follow the number.

'+' Specifies that '+' or '-' is to be printed, as appropriate.

If the '+' appears at the beginning of the format specification, the sign is printed immediately before the number (and any leading dollar).

If the '+' appears at the end of the format specification, the sign is printed after the number (and any exponent part).

'-' May only appear at the end of a format specification. Causes '-' to be printed if the number is negative, and a space to be printed if it is positive. This sign indication is printed after the number (and any exponent part).

Exponent Option:

'↑↑↑↑' following the body of the number, and preceding any trailing sign indication, enables the exponent option. The four characters reserve space for the four character exponent part.

The body of the number is printed with the maximum possible number of digits before the decimal point, reserving one digit position for the sign (even if the number is positive) if no other provision is made for it.

Field Overflow:

If a number cannot be printed within the format given, BASIC attempts to get as close as it can to the required format, but precedes the result by a '%' to indicate field overflow.

No number whose absolute value is greater than or equal to 1E9 can be printed in unscaled format. Any attempt to do so causes the number to be printed in free format preceded by '%' to indicate a format failure.

4. Print Continuation.

Terminating the PRINT command with a separator, TAB or SPC stops it from issuing a newline, and causes the next PRINT command to be treated as a continuation of this one.

Associated Keywords.

WIDTH, ZONE

RAD

Set Radians Mode. **Command**

Use.

The functions SIN, COS, TAN and ATN can work either in degrees or in radians. By default they work in radians, but this may be changed by the DEG command. RAD sets the functions back to working in radians.

Form.

RAD

Notes.

Degrees mode persists after a DEG command until a RAD or CLEAR command is executed, or a new program is LOADed or RUN.

Associated Keywords.

DEG, SIN, COS, TAN, ATN

RANDOMIZE

Randomise the current random number seed. **Command**

Use.

BASIC's random number generator produces a pseudo random sequence, in which the each number depends completely on the previous number. Starting from a given value the sequence is always the same. `RANDOMIZE` sets a new initial value for the random number generator, either to a given value, or to a value entered by the operator.

Form.

`RANDOMIZE` [‹numeric expression›]

If the expression is omitted then the operator is prompted :

```
Random Number Seed ?
```

to get a suitable value, which may be any form of number.

Notes.

Setting the initial random number to the same number will result in the same sequence being generated.

`RANDOMIZE TIME` will produce a sequence which would be hard to repeat.

Associated Keywords.

`RND`

READ

Read from DATA statements. **Command**

Use.

READ fetches data from DATA statements and assigns it to variables.

Form.

READ ‹list of: ‹variable›

Notes.

All the DATA statements in a program, taken in order, form a 'file' of constants. These constants may be read into variables by the READ command. READ steps on to the next constant in the list and assigns it to the next variable in the list of variables given. The constant and the variable to which it is read must be compatible. Attempting to READ past the last DATA constant generates an error (Error 4).

Associated Keywords

DATA, RESTORE

RELEASE

Release sound channels. **Command**

Use.

When a sound is placed on a sound queue it may include a 'hold' state. If any of the channels specified in this command are in 'hold' state, then they are released.

Form.

`RELEASE` ‹sound channels›

‹sound channels› is an ‹integer expression› yielding a value in the range 1..7 which specifies which channels are to be released. This value is 'bit significant' where :

`Bit 0` (the least significant bit) specifies channel A
`Bit 1` specifies channel B
and `Bit 2` specifies channel C

Notes.

The `RELEASE` command has no effect on channels which are not 'held'.

Associated Keywords.

`SOUND`

REM

Remark. **Command**

Use.

To put comments into BASIC programs.

Form.

`REM` ‹rest of line›

BASIC ignores the rest of the line after the `REM` command. Note that colon (statement separator) is also ignored.

Notes.

It is permissable to transfer control to a `REM` command, by `GOTO` or by `GOSUB`.

A single quote character in a line (not in a string) is equivalent to `:REM`, and may also prefix comment text. The one exception to this is in a `DATA` command, where a single quote is treated as part of an unquoted string.

REMAIN

Get delay timer remaining count. **Function**

Use.

To read the remaining count from a given delay timer, and disable it.

Form.

`REMAIN (‹timer number›)`

The ‹timer number› is an ‹integer expression› which specifies which of the four delay timers to read from. The expression must yield a value in the range 0..3.

Notes.

The function returns the remaining count. Zero is returned if the timer was not enabled.

Associated Keywords.

`AFTER, EVERY`

RENUM

Renumber the current program. **Command**

Use.

To renumber all or part of a program.

Form.

RENUM [‹new line number›][,[‹old line number›][,‹increment›]]

‹new line number› is a ‹line number›, giving the new number for the first line to be renumbered. If omitted 10 is assumed.

‹old line number› is a ‹line number›, giving the first line to be renumbered. If omitted the first line of the program is assumed.

‹increment› is a ‹line number›, giving the increment to use between the new line numbers. If omitted 10 is assumed.

Note that all the parameters may be omitted, and this is equivalent to RENUM 10,,10.

Notes.

RENUM adjusts references to line numbers throughout the program to refer to the changed line numbers. Line number references in the following commands are affected :

DELETE, the DELETE in CHAIN MERGE, ELSE, GOSUB, GOTO, LIST, RESTORE, RESUME, RUN, THEN and the various ON....GOTO and ON....GOSUB.
(see also ERL which interacts with RENUM).

When part of the program is renumbered RENUM disallows any attempt to change the order of lines, or to use line numbers already in use in the part of the program which is not being renumbered.

If RENUM discovers line number references to lines that do not exist BASIC produces the message :

```
Undefined line 99999 in 88888
```

where: 99999 is the line number which is referred to, but does not exist, the reference is left unchanged.
and: 88888 is the new number of the line in which the reference occurs.

RESTORE

Restore pointer into DATA list. **Command**

Use.

To move the internal pointer to the next DATA item to a given position (typically back to the first DATA statement).

Form.

RESTORE [‹line number›]

The ‹line number› gives the line to which the DATA pointer is to be set. If omitted the pointer is set back to the start of the program.

Notes.

READ commands move a pointer to DATA items through the program. The RESTORE command moves that pointer to a specified position in the program. The next READ command executed will search forward from that point for the next DATA statement, and start reading from there.

Associated Keywords.

READ, DATA

RESUME

Resume execution after processing an error.

Command
Legal only in Error Processing Mode

Use.

When an error has been trapped by an `ON ERROR GOTO` routine, and has been processed by it, `RESUME` allows normal execution to continue, from a variety of points.

Forms.

```
   RESUME
or RESUME ‹line number›
or RESUME NEXT
```

Notes.

`RESUME` is legal only during Error Processing Mode (ie in an `ON ERROR GOTO` routine).

`RESUME` with no parameter, returns control to the beginning of the statement in which the original error was detected.

`RESUME` ‹line number› returns control to the specified line.

`RESUME NEXT` returns control to the statement immediately after the statement in which the original error was detected.

Associated Keywords.

`ON ERROR GOTO`

RETURN

Return from subroutine. **Command**

Use.

Signals the end of a subroutine. BASIC returns to continue processing after the GOSUB which invoked it.

Form.

```
RETURN
```

Notes.

RETURN is used at the end of normal subroutines and subroutines invoked by 'interrupts'.

Associated Keywords.

GOSUB, ON x GOSUB, ON SQ GOSUB, AFTER n GOSUB, EVERY n GOSUB, ON BREAK GOSUB

RIGHT$

Extract right hand part of string. **Function**

Use.

To extract a given number of characters from the right hand end of a given string.

Form.

`RIGHT$(`‹string expression›`,`‹required length›`)`

The ‹string expression› gives the string from which to extract characters.

The ‹required length› is an ‹integer expression› giving how many characters to extract. The expression must yield a value in the range 0..255.

Notes.

If the given string is shorter than the ‹required length›, then the entire string is returned. Otherwise the right hand end of the string is returned, giving a string of the ‹required length›.

Associated Keywords.

`MID$, LEFT$`

RND

Random Number. **Function**

Use.

To get a random number. This may be the next in the current sequence, the last random number repeated or the first in a new sequence.

Form.

`RND[(`‹numeric expression›`)]`

Notes.

`RND` returns a Real value, where $0 \leq$ value < 1. The pseudo random number generator produces each number by operating on the previous one, so that from the same initial value, the same sequence is produced.

`RND` with the argument omitted, or with an expression yielding a value greater than zero, returns the next random number in the current sequence.

`RND` with an expression yielding the value zero returns a copy of the last random number generated.

`RND` with an expression yielding a value less than zero starts a new sequence loosely, but predictably, based on that value. The first number in the new sequence is returned.

Associated Keywords.

`RANDOMIZE`

ROUND

Round value. **Function**

Use.

To round a value to a given number of decimal places, or to a given power of ten.

Form.

`ROUND(`‹numeric expression›`[,`‹decimals›`])`

The ‹numeric expression› yields the value to be rounded, which may be of any numeric type.

The optional ‹decimals› parameter specifies the number of decimal places to round to. If present ‹decimals› is an ‹integer expression› yielding a value in the range −39..+39. If the ‹decimals› parameter is omitted zero is assumed.

Notes.

The value of the ‹numeric expression› is rounded to the number of decimal places specified by ‹decimals›, as follows :

‹decimals› greater than zero
: the value is rounded to the given number of digits after the decimal point.

‹decimals› zero or absent
: the value is rounded to an integer

‹decimals› less than zero
: the value is rounded to give `ABS(`‹decimals›`)` zeros before the decimal point.

(Rounding means round to nearest, as described in 2.9.1.)

Associated Keywords.

`INT, FIX, CINT`

RUN ‹filename›

Load and start executing a program. **Command**

Use.

To load a program from cassette and start executing it.

Form.

`RUN` ‹filename›

The ‹filename› is a ‹string expression› giving the filename of the file to load from.

Notes.

Any existing program, user functions, variables and arrays are deleted from memory. `DEFINT`, `DEFREAL` and `DEFSTR` settings are reset. All cassette files are abandoned - any buffered output is lost.

`RUN` ‹filename› is equivalent to `LOAD` ‹filename› followed by `RUN` (except in the case of protected BASIC programs, where `LOAD` followed by `RUN` does not work, but `RUN` ‹filename› does).

If the filename ‹string expression› yields an empty string, then BASIC attempts to load and run the first file encountered on the tape.

If the first character of the filename ‹string expression› is '!', then it is removed from the filename and has the effect of suppressing the messages generated by the cassette reading software.

Associated Keywords:

`LOAD, SAVE, CHAIN`

RUN [‹line number›]

Start executing current program. **Command**

Use.

To start executing the current program, either at the beginning, or at a given line.

Form.

RUN [‹line number›]

If the ‹line number› is omitted, then execution starts at the first line of the program.

Notes.

Any existing user functions, variables and arrays are deleted from memory. DEFINT, DEFREAL and DEFSTR settings are reset. All cassette files are abandoned - any buffered output is lost.

SAVE

Save program or memory on cassette. **Command**

Use.

To write the program currently in memory to cassette, or to write an area of machine memory to a file.

Form.

SAVE ‹file name›[,‹file type›[,‹binary parameters›]]

The ‹file name› is a ‹string expression› giving the name of the file to which the program is to be written.

The ‹file type›, if present, defines what sort of file is to be created. If no ‹file type› is given the current BASIC program is saved on tape in BASIC internal form. The ‹file type› is given as a single letter, which may be one of:

A - the current BASIC program is saved as a text file (ASCII).
P - the current BASIC program is saved in protected internal form.
B - save an area of memory as a Binary file.

The ‹binary parameters› are required for ‹file type› B, but are otherwise illegal. See below for a discussion of Binary SAVE.

Notes.

If the first character of the filename ‹string expression› is '!', then it is removed from the filename and has the effect of suppressing the messages generated by the cassette writing software.

After a SAVE operation CAT may be used to verify that the file written is readable.

SAVE without a ‹file type› creates an unprotected BASIC program file.

SAVE ,A creates an ASCII text file in the same way as a LIST #9 would.

SAVE ,P creates a protected BASIC program file. Protected programs may only be RUN or CHAINed. If BASIC returns to Direct Mode for any reason while the current program is protected, then a NEW is performed.

SAVE ,B creates a Binary file containing a copy of the area of memory specified in the ‹binary parameters›, which take the form:

‹start address› , ‹length›[, ‹entry point›]

where :

‹start address› is an ‹address expression› giving the address of the first byte to be written.

‹length› is an ‹address expression› giving how many bytes are to be written.

‹entry point› is an ‹address expression› giving the address at which to start execution if the file is RUN as a program. If the ‹entry point› parameter is omitted it is set to zero - which resets the system if the file is RUN.

The start, length and entry point are recorded with the data, and used by RUN or LOAD. Binary SAVE Binary saves the contents of the machine's read/write memory, so can save from the screen memory under the 'upper ROM' and the memory under the 'lower ROM'.

Associated Keywords.

LOAD, RUN, MERGE, CHAIN, CHAIN MERGE

SGN

Sign of value. **Function**

Use.

To determine the sign of a given value.

Form.

SGN(‹numeric expression›)

Notes.

SGN returns an Integer value :

−1	if ‹numeric expression›	is < 0
Ø	if ‹numeric expression›	is = 0
+1	if ‹numeric expression›	is > 0

Associated Keywords.

ABS

SIN

Sine. **Function**

Use.

To calculate the Sine of a given angle. In radians mode the angle is expressed in radians. In degrees mode the angle is expressed in degrees.

Form.

`SIN(`‹numeric expression›`)`

The ‹numeric expression› gives the angle in radians or degrees. The value of the argument, when converted to radians, must yield a value in the approximate range −200,000..200,000.

Notes.

With angles very much greater than 2*PI (360 degrees) the accuracy of this function becomes increasingly badly affected by the scaling of the angle into the range −PI..+PI (−180..+180 degrees). Rather than return a very inaccurate figure, BASIC will not evaluate `SIN` for values much beyond the range given above.

Associated Keywords.

`COS, TAN, ATN`

SOUND

Put a sound command into the sound queue. **Command**

Use.

The system maintains a separate queue of commands for each of the three sound channels. The `SOUND` command specifies a sound to be generated, and inserts it into the required queue or queues.

Form.

`SOUND` ‹channel status›,‹tone period›[,‹duration›[,‹volume› [,‹volume envelope› [,‹tone envelope›[,‹noise period›]]]]]

The first two parameters are mandatory, the rest are all optional. The parameters are as follows:

‹channel status›: an ‹integer expression› in the range 1..255

The value is 'bit significant' as follows :

`Bit 0` : send sound to channel A
`Bit 1` : send sound to channel B
`Bit 2` : send sound to channel C

`Bit 3` : rendezvous with channel A
`Bit 4` : rendezvous with channel B
`Bit 5` : rendezvous with channel C

`Bit 6` : hold

`Bit 7` : flush

‹tone period›: an ‹integer expression› in the range 0..4095

Sets the tone. A tone period `P` produces a tone of frequency `F`, where `F=125000/P`.

A ‹tone period› of `0` produces no tone at all, which is useful when the sound is to contain only noise.

‹duration›: an ‹integer expression›

The effect of the ‹duration› parameter depends on its value :

`> 0` specifies the duration of the sound in 1/100ths of a second.

`= 0` specifies that the sound should last until the end of the volume envelope.

`< 0` specifies that the volume envelope should be repeated `ABS(`‹duration›`)` times.

If no ‹duration› is specified 20 is assumed, so the sound will last one fifth of a second.

‹volume›: an ‹integer expression› in the range 0..15.

Specifies the initial volume for the sound. If a volume envelope is specified then that may vary the volume during the sound.

If no ‹volume› is specified 12 is assumed.

‹volume envelope›: an ‹integer expression› in the range 0..15

Specifies which volume envelope, if any, to use while generating the sound.

If no ‹volume envelope› is specified, or if the envelope specified is not set then envelope 0 is used.

Envelope 0 is a constant, and cannot be changed by the ENV command. The envelope leaves the volume at the ‹volume› level for 2 seconds.

‹tone envelope›: an ‹integer expression› in the range 0..15.

Specifies which tone envelope, if any, to use while generating the sound.

If envelope 0 or no ‹tone envelope› is specified, or if the envelope specified is not set, then none is used and the tone remains constant (or absent if the tone period is zero).

‹noise period›: an ‹integer expression› in the range 0..31.

Specifies what, if any, noise to add to the sound. Omitting the parameter or using zero means add no noise.

Notes.

The SOUND command checks that there is room in the queue of each of the affected channels before inserting it in any of the queues. SOUND will wait indefinitely for room to be made - by active sounds finishing.

SOUND disables any ON SQ interrupt enabled for each of the affected channels.

The Flush bit in the ‹channel status› causes all the affected channels to be reset. The current active sound, if any, is terminated, and the queue emptied. The new sound is then inserted in the required queues and will start executing, unless prevented by Hold or Rendezvous requirements..

The Hold bit in the ‹channel status› is added to the entry in the sound queue. When a sound with Hold set reaches the head of its queue that channel will stop generating sound and remain held until a suitable RELEASE command is issued. This mechanism is intended to allow sound queues to be primed, with the first entries Held.

The Rendezvous is a mechanism for forcing separate channels into step. A sound on channel A marked to rendezvous with channel B will not start until a sound on channel B marked to rendezvous with channel A is also ready to start - and vice versa. All three channels may be marked to rendezvous with each other.

If more than one channel is specified in the ‹channel status› the sounds are implicitly set to Rendezvous.

The sound generation hardware has one noise period setting for all three channels,so each noise period setting overrides the previous one.

Associated Keywords.

ENV, ENT, RELEASE, ON SQ, SQ

SPACE$

String of spaces. **Function**

Use.

To create a string of spaces of a given length.

Form.

`SPACE$(‹length›)`

The ‹length› is an ‹integer expression› which specifies the required string length. The expression must yield a value in the range 0..255. (A ‹length› of zero produces a null string.)

Associated Keywords.

`SPC, TAB, STRING$`

SPEED INK

Set the periods for flashing Inks. **Command**

Use.

The INK and BORDER commands allow two colours to be associated with each Ink, in which case the Ink alternates between the two colours. This command sets the times between colour changes.

Form.

SPEED INK ‹period› , ‹period›

The two ‹periods› are ‹integer expression›s which must yield values in the range 1..255. The first specifies the time for which the Ink's first colour is to be used, the second specifies the time for the Ink's second colour.

Notes.

To avoid unpleasant visual effects the palette is changed during frame fly back. The times are, therefore, given in frame periods - which are 1/50th's or 1/60th's of a second, depending on the TV standard in use. In the UK 1/50ths are used. This is the only timing which varies between versions of the machine.

Associated Keywords.

INK, BORDER

SPEED KEY

Set the keyboard repeat speed. **Command**

Use.

Pressing a key causes one key value to be generated, if the key is held down it will, after a time, repeat. This command sets the delay before the key repeats and the rate at which it then repeats.

Form.

SPEED KEY ‹start delay› , ‹repeat period›

The ‹start delay› is an ‹integer expression› in the range 1..255, specifying in 1/50th's of a second the delay between the key first being pressed and the first repeated value.

The ‹repeat period› is an ‹integer expression› in the range 1..255, specifying in 1/50th's of a second the time between each repeated value, once repeating has started.

Notes.

Setting a very small ‹start delay› can make the keyboard difficult to use because the repeat action starts to interact with the keyboard scanning.

Not all keys repeat. The KEY DEF command allows repeating to be enabled and disabled for each key.

Associated Keywords.

KEY DEF

SPEED WRITE

Set the Cassette Write Speed. **Command**

Use.

Data may be written to Cassette at one of two speeds. This command selects which to use.

Form.

`SPEED WRITE` ‹integer expression›

The ‹integer expression› must yield a value in the range 0..1, specifying which speed to use, thus :

0 : Nominal 1000 bits per second
1 : Nominal 2000 bits per second

Notes.

The speeds given are nominal because they assume that on average an equal number of zeros and ones are written, and because no account is taken of the 'red tape' around each block.

Two speeds are supported because cassette tape is not the most reliable of media. While the higher speed should give adequate performance, the lower speed has a greater safety margin.

The machine defaults to the lower writing speed.

When reading data from tape the machine uses the leading synchronisation bits on each block to adapt itself to the speed.

SQ

Test state of Sound Queue. **Function**

Use.

The SQ function may be used to find how many free entries there are in the queue for a given channel, whether the channel is active and if not why the entry at the head of the queue (if any) is waiting.

Form.

SQ(‹channel›)

The ‹channel› is an ‹integer expression› yielding one of the values :

1 : for channel A
2 : for channel B
4 : for channel C

Notes.

The SQ function returns an Integer value which comprises the following bit fields :

Bits 0..2	the number of free entries in the queue, in the range 0..4	
Bits 3..5	the Rendezvous state of the head of the queue (if any).	mutually exclusive
Bit 6	is set if the head of the queue is Held.	
Bit 7	set if the channel is currently active.	

The Rendezvous and Held states are described in the SOUND command.

The SQ function disables any ON SQ interrupt set for the channel.

Associated Keywords.

SOUND, ON SQ GOSUB

SQR

Square Root. **Function**

Use.

Evaluate the square root of a given value.

Form.

SQR(‹numeric expression›)

Where the ‹numeric expression› must yield a positive value.

STOP

Stop execution. **Command**

Use.

To stop execution of a program, but leave BASIC in a state where the program can be restarted after the `STOP` command. This may be used to stop a program at a particular point when debugging.

Form.

`STOP`

Notes.

When the `STOP` is obeyed BASIC halts program execution and prints a '`Break`' message.

Execution may be restarted after a `STOP` by using the `CONT` command, provided that the program has not been altered in any way in the interim.

Associated Keywords.

`CONT, END`

STR$

String representation of numeric value. **Function**

Use.

To convert given numeric value to a decimal string representation.

Form.

STR$(‹numeric expression›)

Notes.

The value of the ‹numeric expression› is converted to a decimal string in the same form as used in a PRINT command. Note that positive values yield a string with a leading space, where negative values have a leading minus sign.

Associated Keywords.

VAL, PRINT, DEC$, HEX$, BIN$

STRING$

String of a particular character. **Function**

Use.

To construct a string consisting of a given character repeated a given number of times. The character may be specified by its numeric value.

Form.

`STRING$(`‹length›`,`‹character specifier›`)`

The ‹length› is an ‹integer expression› giving the required length of the result string. The expression must yield a value in the range 0..255.

The ‹character specifier› may be one of:

‹integer expression› specifying `CHR$(`‹integer expression›`)`
‹string expression› specifying the first character of the string

Associated Keywords.

`SPACE$`

SYMBOL

Define matrix for User Defined Symbol. **Command**

Use.

The SYMBOL command redefines the representation of a given character.

Form.

SYMBOL ‹character number› , ‹list of: ‹row›

The ‹character number› is an ‹integer expression› in the range n..255, where n is the first User Definable Symbol (see SYMBOL AFTER).

The ‹list of: ‹row› is a list of between one and eight ‹integer expression›s, each of which must yield a value in the range 0..255. If less than eight rows are specified then the unspecified rows are set to zero.

Notes.

The matrix for a character is an eight byte area, each bit of which corresponds to a pixel in the eight by eight screen character cell. When the character is written to the screen a zero bit in the matrix is sets the corresponding pixel to the current Paper ink, unless Transparent mode is enabled, when the correponding pixel is unaffected. A one bit in the matrix produces a pixel in the Pen ink.

The rows specified in the SYMBOL command are written to the character matrix in the order given. The most significant bit of the first row corresponds to the top left hand pixel of the screen character cell. The least significant bit of the last row corresponds to the bottom right hand pixel of the screen character cell.

Changing the matrix for a character does not have any effect on characters already displayed.

The User Defined Symbols are reset to their initial values each time a SYMBOL AFTER command is issued.

Associated Keywords.

SYMBOL AFTER

SYMBOL AFTER

Define a new number of User Defined Symbols. **Command**

Use.

The number of characters whose screen representation may be altered by the SYMBOL command is set by the SYMBOL AFTER command.

Form.

SYMBOL AFTER ‹integer expression›

The ‹integer expression› must yield a value in the range 0..256, and specifies the number of the first character which is to be user definable.

Notes.

When the machine is first turned on an implicit SYMBOL AFTER 240 is obeyed, giving 16 user definable characters.

SYMBOL AFTER n sets all characters from n to 255 inclusive to be user definable. SYMBOL AFTER 256 sets no characters to be user definable. SYMBOL AFTER Ø sets all characters to be user definable.

Whenever a SYMBOL AFTER command is obeyed all user defined characters are reset to their original state.

SYMBOL AFTER commands are not allowed if HIMEM has been altered since the last SYMBOL AFTER command (unless the last one was SYMBOL AFTER 256). HIMEM is altered by MEMORY commands and by the automatic allocation of cassette buffers. See Appendix IX for a fuller description of the interactions.

Associated Keywords.

SYMBOL

TAG

Text At Graphics Cursor. **Command**

Use.

Text sent to a given stream may be redirected to be written at the graphics cursor position. This allows text and symbols to be mixed with graphics.

Form.

TAG [#‹stream expression›]

The ‹stream expression› must yield a screen stream. If it is omitted stream #0 is assumed.

Notes.

A character written at the graphics cursor is written so that the top left hand corner of the character cell is at the graphics cursor position.

The graphics cursor is advanced by eight pixels after each character written this way. The number of horizontal graphics increments represented by eight pixels varies from mode to mode.

Control characters are not recognised when sent to the graphics screen, and only have the effect of displaying the symbol defined for that character value. Note in particular that carriage returns and line feeds, including those produced automatically by BASIC, do not move the graphics cursor down and left, but produce one or two symbols on the screen instead.

Stream #0 is forced out of TAG state when BASIC returns to Direct Mode.

Associated Keywords.

TAGOFF, SYMBOL

TAGOFF

Text At Graphics Cursor Off. **Command**

Use.

Cancels TAG for a given stream.

Form.

TAGOFF [#‹stream expression›]

The ‹stream expression› must yield a screen stream. If it is omitted stream #0 is assumed.

Notes.

Text sent to the given stream is again directed to the text cursor position. The text cursor is not moved by characters sent to the stream while TAG is in force.

Associated Keywords.

TAG

TAN

Tangent. **Function**

Use.

To calculate the Tangent of a given angle. In radians mode the angle is expressed in radians. In degrees mode the angle is expressed in degrees.

Form.

`TAN (`‹numeric expression›`)`

The ‹numeric expression› gives the angle in radians, or degrees. The value of the argument, when converted to radians must be in the approximate range −200,000..200,000.

Notes.

With angles very much greater than 2*PI (360 degrees) the accuracy of this function becomes increasingly badly affected by the scaling of the angle into the range −PI....+PI (−180....+180 degrees). Rather than return a very inaccurate figure, BASIC will not evaluate `TAN` for values much beyond the range given above.

Associated Keywords.

`COS, SIN, ATN`

TEST

Test ink used at given graphics position. **Function**

Use.

TEST may be used to read back the ink used at a given position on the screen.

Form.

TEST (‹x coordinate› , ‹y coordinate›)

The ‹x coordinate› and ‹y coordinate› must be ‹integer expression›s.

Notes.

The TEST function returns the ink in use at the graphics screen position specified by the coordinates given. If the position specified is outside the current graphics window, then the ink used in the latest CLG command is returned - if no CLG command has been executed then Ø is returned.

TEST moves the graphics cursor to the position tested.

Associated Keywords.

TESTR, MOVE, MOVR, PLOT, PLOTR, DRAW, DRAWR

TESTR

Test ink used at given graphics position. **Function**

Use.

TESTR may be used to read back the ink used at a given position on the screen, where that position is specified relative to the current graphics cursor position.

Form.

TESTR (‹x offset›,‹y offset›)

The ‹x offset› and ‹y offset› must be ‹integer expression›s.

Notes.

The TESTR function returns the ink in use at the graphics screen position specified by adding the ‹x offset› to the current x coordinate and the ‹y offset› to the current y coordinate. If the position specified is outside the current graphics window, then the ink used in the latest CLG command is returned - if no CLG command has been executed then 0 is returned.

TESTR moves the graphics cursor to the position tested.

Associated Keywords.

TEST, MOVE, MOVR, PLOT, PLOTR, DRAW, DRAWR

TIME

Elapsed time. **Function**

Use.

A count of the time elapsed since the machine was turned on is maintained. The `TIME` function returns its value.

Form.

```
TIME
```

Notes.

The time is kept in units of 1/300th's of a second.

The time is not kept up to date while reading and writing the cassette.

TRON
TROFF

Turn on and off tracing. **Command**

Use.

BASIC includes a facility to trace the execution of a program, by reporting the number of each line just before it is executed.

Form.

```
TRON
TROFF
```

Notes.

`TRON` enables the trace feature. `TROFF` disables it.

While tracing is enabled the number of each line is printed on the console (enclosed in square brackets) just before the line is executed.

`TRON` is automatically reset whenever a new program is loaded, that is by `NEW`, `LOAD`, `CHAIN` and `RUN` ‹filename› commands.

UNT

Convert unsigned integer. **Function**

Use.

To convert the given unsigned value to an integer.

Form.

UNT (‹address expression›)

Notes.

UNT returns an Integer value in the range −32,768..32767 (ie the twos complement equivalent of the unsigned value of the ‹address expression›) - see section 2.10.

Associated Keywords.

INT, FIX, CINT, ROUND

UPPER$

Convert string to upper case. **Function**

Use.

To create a new string which is a copy of another, with all lower case alphabetic characters converted to their upper case equivalents.

Form.

`UPPER$(`‹string expression›`)`

Notes.

The result of the ‹string expression› is returned, with any characters in the range 'a'..'z' converted to the equivalent character in the range 'A'..'Z'.

Associated Keywords.

`LOWER$`

VAL

Convert string to numeric value. **Function**

Use.

To take a string representation of a number and convert it to a numeric value.

Form.

VAL(‹string expression›)

Notes.

VAL converts the given string to a number in the same way as INPUT.

Associated Keywords.

STR$

VPOS

Find current Vertical Position. **Function**

Use.

To establish the current vertical position on a given stream.

Form.

`VPOS(#‹stream expression›)`

The ‹stream expression› must yield a screen stream.

Notes.

`VPOS` returns the vertical position of the cursor in current window for the given stream. The top left hand corner of a window is vertical position 1.

Associated Keywords.

`POS`

WAIT

Wait on I/O port. **Command**

Use.

To wait until a given I/O port returns a particular value.

Form.

WAIT ‹port number› , ‹mask›[, ‹inversion›]

The ‹port number› is an ‹address expression›.

Both the ‹mask› and ‹inversion› parts are ‹integer expression›'s, which must yield values in the range 0..255.

Notes.

BASIC enters a loop reading from the I/O port, Exclusive ORing the value read with the ‹inversion› and then ANDing with the ‹mask›, until the result is non-zero. (If no ‹inversion› is given, then the Exclusive OR step is omitted.).

Note that there is no way of interrupting this loop, so BASIC will remain in it indefinitely if the required condition never occurs.

Associated Keywords.

INP, OUT

WEND

Delimit end of a WHILE loop. **Command**

Use.

A WHILE loop repeatedly executes a body of program until a given condition is true. The WEND command defines the end of the loop.

Form.

WEND

Notes.

The WEND command defines the end of a WHILE loop. The way in which WHILE and WEND are tied together is described under WHILE. When a WEND command is encountered BASIC knows which (if any) WHILE it must be associated with.

Associated Keywords.

WHILE

WHILE

WHILE loop. **Command**

Use.

A WHILE loop repeatedly executes a body of program until a given condition is true. The WHILE command defines the head of the loop, and gives the condition.

Form.

WHILE ‹logical expression›

Notes.

When the WHILE command is obeyed the ‹logical expression› is evaluated. If the result is zero BASIC skips to the matching WEND. If the result is not zero, execution continues until the matching WEND is met, whereupon BASIC loops back to the WHILE command, and the process is repeated.

The WEND command which matches a given WHILE is established statically when the WHILE is first executed. That is to say that the WEND which matches the WHILE depends on the order of statements in the program, quite independently of the order of execution. It is not possible, therefore, to have more than one WEND associated with a given WHILE.

WHILE loops may be nested.

It is permissable to terminate a WHILE loop by avoiding the WEND.

Associated Keywords.

WEND, FOR

WIDTH

Set width of printer. **Command**

Use.

To tell BASIC how wide the printer is in characters. This information is used when printing, to allow BASIC to insert carriage returns at the appropriate moments.

Form.

`WIDTH` ‹integer expression›

The ‹integer expression› gives the printer width, and must yield a value in the range 1..255 (though small values may have curious effects).

Notes.

The initial value of the printer width is 132.

Setting the width to 255 has special meaning. BASIC treats the printer as being infinitely wide, so never inserts carriage returns.

BASIC maintains a count of printing characters sent to the printer since the last carriage return, giving the logical position across the printer. When the logical position reaches 255 the counter is no longer incremented, so all positions greater than 255 are treated as 255. (Printing characters are those with values greater than 31 (&1F).)

Associated Keywords.

`PRINT, POS`

WINDOW

Set text window on screen. **Command**

Use.

To set the text window for a given screen stream.

Form.

`WINDOW` [#‹stream expression›,]‹left›,‹right›,‹top›,‹bottom›

The ‹stream expression› specifies which stream's window to set. If it is omitted, stream `#0` is assumed.

The other parameters must all be ‹integer expression›s yielding values in the range 1..255, and specify the edge positions of the window in absolute screen row and column positions, where the top left hand corner of the screen is position 1,1.

‹left›	The leftmost column to be included in the window.
‹right›	The rightmost column to be included in the window.
‹top›	The highest (lowest numbered) row to be included in the window.
‹bottom›	The lowest (highest numbered) row to be included in the window.

Notes.

The cursor for the given stream is moved to the top left hand corner of the new window, position 1,1.

The order of the ‹left› and ‹right› parameters may be reversed, since ‹right› must be greater than or equal to ‹left›.

The order of the ‹top› and ‹bottom› parameters may be reversed, since ‹bottom› must be greater than or equal to ‹top›.

Associated Keywords.

`ORIGIN`

WINDOW SWAP

Swap text windows. **Command**

Use.

If no stream is specified in commands which take a stream parameter then stream #0 is used. Messages produced by BASIC are sent to stream #0. This command allows, amongst other effects, a stream to be exchanged with stream #0 to 'redirect' text sent to stream #0

Form.

`WINDOW SWAP` ‹stream expression› , ‹stream expression›

Where both ‹stream expression›s must specify a screen stream.

Notes.

All the attributes of the text streams are exchanged, including the Window, Cursor Position, Pen, Paper and Tag attributes.

Associated Keywords.

`WINDOW, PEN, PAPER, TAG`

WRITE

Write to stream. **Command**

Use.

To print the values of a number of expressions to the given stream, separating them by commas and enclosing strings in double quotes.

Form.

`WRITE` [#‹stream expression›,][‹write list›]

The ‹stream expression› specifies which stream to write to. If it is omitted, stream `#0` is assumed.

‹write list› is: ‹expression›[‹separator›‹expression›]*

where ‹separator› may be comma or semi-colon, interchangeably.

Notes.

`WRITE` is similar to `PRINT`, except that :

- print zones are ignored
- strings are printed enclosed in double quotes
- commas are added between the printed items
- `WRITE` does not support the trailing separator option

`WRITE` is intended for use with the cassette stream. The form in which the data is written allows it to be read back straightforwardly using `INPUT`.

Associated Keywords.

`PRINT`

XPOS

X Graphics Position. **Function**

Use.

To establish the X (horizontal) position of the graphics cursor.

Form.

`XPOS`

Notes.

`XPOS` returns an Integer value giving the current graphics cursor X position.

Associated Keywords.

`YPOS, MOVE, MOVER, ORIGIN`

YPOS

Y Graphics Position. **Function**

Use.

To establish the Y (vertical) position of the graphics cursor.

Form.

`YPOS`

Notes.

`YPOS` returns an Integer value giving the current graphics cursor Y position.

Associated Keywords.

`XPOS, MOVE, MOVER, ORIGIN`

ZONE

Set Print Zone Size. **Command**

Use.

To change the width of the Print Zone used in PRINT.

Form.

ZONE ‹integer expression›

The ‹integer expression› gives the new Print Zone width, and must yield a value in the range 1..255.

Notes.

The default Print Zone width is 13. The width is reset to the default value whenever a new program is loaded, that is by NEW, LOAD, CHAIN and RUN ‹filename› commands.

Associated Keywords.

PRINT, WIDTH

APPENDIX I
Error Numbers and Error Messages.

All errors generated by BASIC are listed here, in error number order. The messages produced by BASIC are given, as well a brief description of possible causes.

1 Unexpected NEXT

A NEXT command has been encountered while not in a FOR loop, or the control variable in the NEXT command does not match that in the FOR.

2 Syntax Error

BASIC cannot understand the given line because a construct within it is not legal.

3 Unexpected RETURN

A RETURN command has been encountered when not in a subroutine.

4 DATA exhausted

A READ command has attempted to read beyond the end of the last DATA.

5 Improper argument

This is a general purpose error. The value of a function's argument, or a command parameter is invalid in some way.

6 Overflow

The result of an arithmetic operation has overflowed. This may be a floating point overflow, in which case some operation has yielded a value greater than 1.7E+38 (approx.). Alternatively, this may be the result of a failed attempt to change a floating point number to a 16 bit signed integer.

7 Memory full

The current program or its variables may be simply too big, or the control structure is too deeply nested (nested GOSUBs, WHILEs or FORs).

A MEMORY command will give this error if an attempt is made to set the top of BASIC's memory too low, or to an impossibly high value. Note that an open cassette file has a buffer allocated to it, and that may restrict the values that MEMORY may use. See Appendix IX.

8 Line does not exist

The line referenced cannot be found.

9 Subscript out of range

One of the subscripts in an array reference is too big or too small.

10 Array already dimensioned

One of the arrays in a DIM statement has already been declared.

11 Division by zero

May occur in Real division, integer division, integer modulus or in exponentiation.

12 Invalid direct command

The last command attempted is not valid in Direct Mode.

13 Type mismatch

A numeric value has been presented where a string value is required, and vice versa, or an invalidly formed number has been found in READ or INPUT.

14 String space full

So many strings have been created that there is no further room available, even after 'garbage collection'.

15 String too long

String exceeds 255 characters in length. May be generated by concatenating strings.

16 String expression too complex

String expressions may generate a number of intermediate string values. When the number of these values exceeds a reasonable limit, BASIC gives up, and this error results.

17 Cannot CONTinue

For one reason or another the current program cannot be restarted using CONT. Note that CONT is intended for restarting after a STOP command, **[ESC][ESC]** or error, and that any alteration of the program in the meantime makes a restart impossible.

18 Unknown user function

No DEF FN has been executed for the FN just invoked.

19 RESUME missing

The end of the program has been encountered while in Error Processing Mode (ie in an ON ERROR GOTO routine).

20 Unexpected RESUME

RESUME is only valid while in Error Processing Mode (ie in an ON ERROR GOTO routine).

21 Direct command found

When loading a program from cassette a line without a line number has been found.

22 Operand missing

BASIC has encountered an incomplete expression.

23 Line too long

A line when converted to BASIC internal form becomes too big.

24 EOF met

An attempt has been made to read past end of file on the cassette input stream.

25 File type error

The cassette file being read is not of a suitable type. OPENIN is only prepared to open ASCII text files. LOAD, RUN et al are only prepared to deal with the file types produced by SAVE.

26 NEXT missing

Cannot find a NEXT to match a FOR command.

27 File already open

An OPENIN or OPENOUT command has been executed before the previously opened file has been closed.

28 Unknown command

BASIC cannot find a taker for an external command.

29 WEND missing

Cannot find a WEND to match a WHILE command.

30 Unexpected WEND

BASIC encountered a WEND when not in a WHILE loop, or a WEND that does not match the current WHILE loop.

APPENDIX II - External Routines and External Commands.

The CALL command may be used to invoke external machine language routines, which must be loaded into machine memory above the highest address used by BASIC. The format of the parameters and the parameter passing mechanism are described here.

External Commands are handled like CALL except that the routine may be in machine memory or in an external ROM, and the address of the routine to call is established dynamically by searching for the command name.

1. Data Formats.

This is a brief description of the format of the three data types. All are held with the least significant byte first (at the lowest address).

1.1 Integer.

Two's complement 16 bit values.

1.2 String.

Strings are held in two parts, the String Descriptor and the String Body. The String Descriptor is a three byte vector. The least significant byte gives the length of the string, zero implies the string is null and that the rest of the descriptor should be ignored. The next two bytes give the machine address of the string body.

1.3 Real.

A four byte mantissa followed by a one byte exponent. Numbers are always held normalised, and the most significant bit of the mantissa is replaced by the sign. The exponent is biased by 128. A zero exponent means that the number is zero, and the mantissa should be ignored. The mantissa is in sign and magnitude form with the binary point to the left of the implied most significant bit.

2. CALLed Subroutines and External Commands.

The CALL command specifies the address of the subroutine, optionally followed by a list of parameters.

An External Command is identified by a vertical bar ("|") followed by the command name, optionally followed by a list of parameters. The command name takes the same form as a BASIC variable name, without any type marker. Note that the vertical bar does not form part of the name, but is simply to distinguish between internal and external commands. BASIC forces all characters in the command name to upper case before searching for an external command routine.

Parameters are passed by value, that is each parameter may be a numeric expression, the result of which is passed, or an address reference. The number and type of parameters must be agreed between the BASIC program and the subroutine - BASIC performs no checking.

Each parameter passed is a two byte number, whose interpretation depends on its type :

Integer expression :	two's complement value of the integer result.
Real expression :	value of the Real result forced to Unsigned Integer.
Address reference :	address of the value of the variable. (Note that the value of a string is its descriptor.)

The entry conditions are :

Register A contains the number of parameters.

Index Register X contains the address of the parameters. If there are n parameters then the ith parameter is at offset (n-i)*2 from the index register address - so the 1st parameter is at the largest offset, and the last parameter is pointed to by X.

The contents of other registers are undefined.

The exit conditions are :

AF' BC' preserved.
All other registers and flags corrupt.

String variables may be altered provided that the String Descriptor is not altered **IN ANY WAY**.

APPENDIX III - BASIC Keywords.

The following are the BASIC keywords, they are reserved and cannot be used as variable names.

```
ABS, AFTER, AND, ASC, ATN, AUTO

BIN$, BORDER

CALL, CAT, CHAIN, CHR$, CINT, CLEAR, CLG,
CLOSEIN, CLOSEOUT, CLS, CONT, COS, CREAL

DATA, DEC$, DEF, DEFINT, DEFREAL, DEFSTR, DEG,
DELETE, DI, DIM, DRAW, DRAWR

EDIT, EI, ELSE, END, ENT, ENV, EOF, ERASE,
ERL, ERR, ERROR, EVERY, EXP

FIX, FN, FOR, FRE

GOSUB, GOTO

HEX$, HIMEM

IF, INK, INKEY, INKEY$, INP, INPUT, INSTR, INT

JOY

KEY

LEFT$, LEN, LET, LINE, LIST, LOAD, LOCATE,
LOG, LOG10, LOWER$

MAX, MEMORY, MERGE, MID$, MIN, MOD, MODE, MOVE, MOVER

NEXT, NEW, NOT
```

ON, ON BREAK, ON ERROR GOTO Ø, ON SQ, OPENIN,
OPENOUT, OR, ORIGIN, OUT

PAPER, PEEK, PEN, PI, PLOT, PLOTR, POKE, POS, PRINT

RAD, RANDOMIZE, READ, RELEASE, REM, REMAIN, RENUM,
RESTORE, RESUME, RETURN, RIGHT$, RND, ROUND, RUN

SAVE, SGN, SIN, SOUND, SPACE$, SPC, SPEED, SQ, SQR,
STEP, STOP, STR$, STRING$, SWAP, SYMBOL

TAB, TAG, TAGOFF, TAN, TEST, TESTR, THEN, TIME, TO, TROFF,
TRON

UNT, UPPER$, USING

VAL, VPOS

WAIT, WEND, WHILE, WIDTH, WINDOW, WRITE

XOR, XPOS

YPOS

ZONE

APPENDIX IV
Text Screen Legal Position & Control Characters

The text cursor may be positioned outside the current window. Various operations force the cursor to a legal position before they are performed, these are :

- writing a character
- drawing the cursor 'blob'
- obeying the control codes marked with an asterisk in the list below

The procedure for forcing the cursor to a legal position is as follows :

a. If the cursor is to the right of the right hand edge, then it is moved to the leftmost column of the next line down.

b. If the cursor is to the left of the left hand edge, then it is moved to the rightmost column of the next line up.

c. If the cursor is now above the top edge, then the window is rolled down a line and the cursor is set to the top line of the window.

d. If the cursor is now below the bottom edge, then the window is rolled up a line and the cursor is set to the bottom line of the window.

The tests and operations are done in the order given. The illegal cursor positions may be zero or negative, which are off to the left or above the window.

Character values in the range 0..31 sent to the text screen do not produce a character on the screen, but are interpreted as control codes. Some of the codes alter the meaning of one or more of the following characters, which are the code's parameters. The codes marked '*' force the cursor to a legal position in the current

window before they are obeyed - but may leave the cursor in an illegal position. The codes and their meanings are as follows :

Value			Name	Parameter	Meaning
&00	0		NUL		No effect Ignored.
&01	1		SOH	0.255	Print the symbol given by the parameter. This allows the symbols in 0..31 to be displayed.
&02	2		STX		Turn off text cursor.
&03	3		ETX		Turn on text cursor. Note that BASIC uses an overriding cursor disable, which is released only when it is awaiting keyboard input.
&04	4		EOT	0..2	Set screen mode. Parameter taken MOD 4. Equivalent to a MODE command.
&05	5		ENQ	0..255	Send the parameter character to the graphics cursor.
&06	6		ACK		Enable Text Screen. (See &15, NAK, below.)
&07	7		BEL		Sound Bleeper. Note that this flushes the sound queues.
&08	8	*	BS		Move cursor back one character.
&09	9	*	TAB		Move cursor forward one character.
&0A	10	*	LF		Move cursor down one line.
&0B	11	*	VT		Move cursor up one line.
&0C	12		FF		Clear text window and move cursor to top left corner. Equivalent to a CLS command.
&0D	13	*	CR		Move cursor to left edge of window on current line.
&0E	14		SO	0..15	Set Paper Ink. Parameter taken MOD 16. Equivalent to PAPER command.
&0F	15		SI	0..15	Set Pen Ink. Parameter taken MOD 16. Equivalent to PEN command.
&10	16	*	DLE		Delete current character. Fills character cell with the current Paper Ink.
&11	17	*	DC1		Clear from left edge of window to, and including, the current character position. Fills affected cells with the current Paper Ink.
&12	18	*	DC2		Clear from, and including, the current character position to the right edge of window. Fills affected cells with the current Paper Ink.

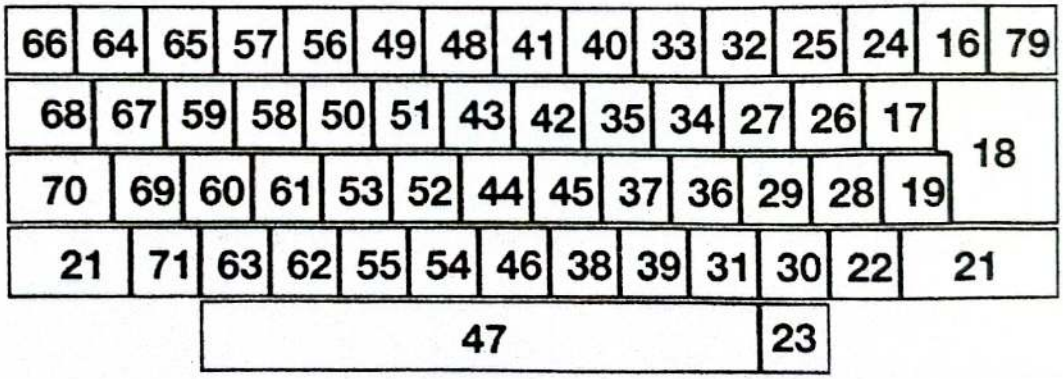
&13	19	*DC3		Clear from start of window to, and including, the current character position. Fills affected cells with the current Paper Ink.
&14	20	*DC4		Clear from, and including, the current character position to the end of window. Fills affected cells with the current Paper Ink.
&15	21	NAK		Turn off text screen. The screen will not react to anything sent to it until after an ACK (&06) is sent.
&16	22	SYN	0..1	Parameter taken MOD 2: 0 disables the transparent option 1 enables the transparent option.
&17	23	ETB	0..3	Parameter MOD 4 0 sets Normal Graphics Ink Mode 1 " XOR " " " 2 " AND " " " 3 " OR " " "
&18	24	CAN		Exchange Pen and Paper Inks.
&19	25	EM	0..255 0..255 0..255 0..255 0..255 0..255 0..255 0..255 0..255	Set Matrix for User Definable Character. Equivalent to a SYMBOL command. Takes nine parameters. The first parameter specifies which character's matrix to set. The next eight specify the matrix : the most significant bit of the first byte corresponds to the top left hand pixel of the character cell, the least significant bit of the last byte corresponds to the bottom right hand pixel of the character cell.
&1A	26	SUB	1..80 1..80 1..25 1..25	Set Window. Equivalent to a WINDOW command. The first two parameters specify the left and right hand edges of the window -the smaller value is taken as the left edge, the larger the right. The second two parameters specify the top and bottom edges of the window - the smaller value is taken as the top edge, the larger the bottom edge.
&1B	27	ESC		No effect. Ignored.
&1C	28	FS	0..15 0..31 0..31	Set Ink to a pair of colours. Equivalent to an Ink command. The first parameter (MOD 16) specifies the Ink, the next two (MOD 32) the required colours.

&1D	29	GS	0..31 0..31	Set Border to a pair of colours. Equivalent to a BORDER command. The two parameters (MOD 32) specify the two colours.
&1E	30	RS		Move cursor to top left hand corner of window.
&1F	31	US	1..80 1..25	Move cursor to the given position in the current window. Equivalent to a LOCATE command.The first parameter gives the column to move to, the second the line.

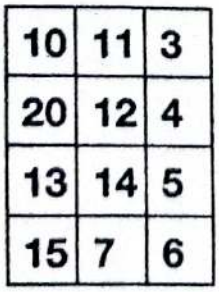
APPENDIX V- Keyboard and Joystick Key Numbers.

The KEY DEF command allows the values produced by a key to be changed, the INKEY function allows the state of an individual key to examined. Each key is refered to by its key number. The decimal key numbers are given in the following diagrams

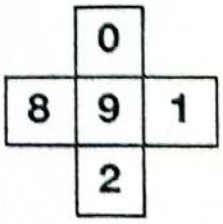
Main Keyboard



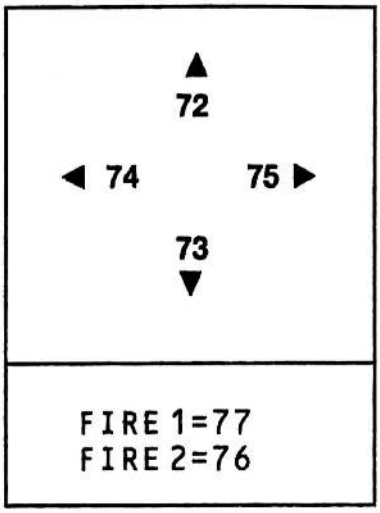
Function/Numeric Keypad



Cursor Keys

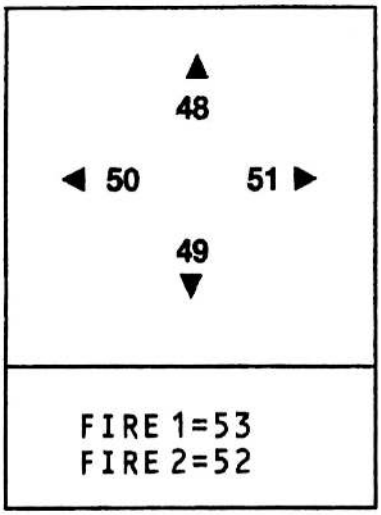


Joystick 0



the second joystick is mapped over some of the keys in the main keyboard, as follows

Joystick 1



APPENDIX VI - Colours.

The colour numbers are arranged so that on monochrome screens the higher the number the brighter the effect. The colour numbers and the colours produced (on a suitable screen) are :

0	BLACK
1	BLUE
2	BRIGHT BLUE
3	RED
4	MAGENTA
5	MAUVE
6	BRIGHT RED
7	PURPLE
8	BRIGHT MAGENTA
9	GREEN
10	CYAN
11	SKY BLUE
12	YELLOW
13	WHITE
14	PASTEL BLUE
15	ORANGE
16	PINK
17	PASTEL MAGENTA
18	BRIGHT GREEN
19	SEA GREEN
20	BRIGHT CYAN
21	LIME
22	PASTEL GREEN
23	PASTEL CYAN
24	BRIGHT YELLOW
25	PASTEL YELLOW
26	BRIGHT WHITE

Appendix VII-Notes and Tone Periods.

The table which follows gives the recommended Tone Period settings for notes in the usual even tempered scale for the full eight octave range.

The frequency produced is not exactly the required frequency because the period setting has to be an integer. The relative error is the ratio of the difference between the required and actual frequencies and the required frequency, ie: (REQUIRED -ACTUAL)/REQUIRED).

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave -3
C	32.703	3822	-0.007%	
C#	34.648	3608	+0.007%	
D	36.708	3405	-0.007%	
D#	38.891	3214	-0.004%	
E	41.203	3034	+0.009%	
F	43.654	2863	-0.016%	
F#	46.249	2703	+0.009%	
G	48.999	2551	-0.002%	
G#	51.913	2408	+0.005%	
A	55.000	2273	+0.012%	
A#	58.270	2145	-0.008%	
B	61.735	2025	+0.011%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave -2
C	65.406	1911	-0.007%	
C#	69.296	1804	+0.007%	
D	73.416	1703	+0.022%	
D#	77.782	1607	-0.004%	
E	82.407	1517	+0.009%	
F	87.307	1432	+0.019%	
F#	92.499	1351	-0.028%	
G	97.999	1276	+0.037%	
G#	103.826	1204	+0.005%	
A	110.000	1136	-0.032%	
A#	116.541	1073	+0.039%	
B	123.471	1012	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave -1
C	130.813	956	+0.046%	
C#	138.591	902	+0.007%	
D	146.832	851	-0.037%	
D#	155.564	804	+0.058%	
E	164.814	758	-0.057%	
F	174.614	716	+0.019%	
F#	184.997	676	+0.046%	
G	195.998	638	+0.037%	
G#	207.652	602	+0.005%	
A	220.000	568	-0.032%	
A#	233.082	536	-0.055%	
B	246.942	506	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave 0
C	261.626	478	+0.046%	*Middle C*
C#	277.183	451	+0.007%	
D	293.665	426	+0.081%	
D#	311.127	402	+0.058%	
E	329.628	379	-0.057%	
F	349.228	358	+0.019%	
F#	369.994	338	+0.046%	
G	391.995	319	+0.037%	
G#	415.305	301	+0.005%	
A	440.000	284	-0.032%	*International A*
A#	466.164	268	-0.055%	
B	493.883	253	-0.038%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave 1
C	523.251	239	+0.046%	
C#	554.365	225	-0.215%	
D	587.330	213	+0.081%	
D#	622.254	201	+0.058%	
E	659.255	190	+0.206%	
F	698.457	179	+0.019%	
F#	739.989	169	+0.046%	
G	783.991	159	-0.277%	
G#	830.609	150	-0.328%	
A	880.000	142	-0.032%	
A#	932.328	134	-0.055%	
B	987.767	127	+0.356%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave 2
C	1046.502	119	-0.374%	
C#	1108.731	113	+0.229%	
D	1174.659	106	-0.390%	
D#	1244.508	100	-0.441%	
E	1318.510	95	+0.206%	
F	1396.913	89	-0.543%	
F#	1479.978	84	-0.548%	
G	1567.982	80	+0.350%	
G#	1661.219	75	-0.328%	
A	1760.000	71	-0.032%	
A#	1864.655	67	-0.055%	
B	1975.533	63	-0.435%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave 3
C	2093.004	60	+0.462%	
C#	2217.461	56	-0.662%	
D	2349.318	53	-0.390%	
D#	2489.016	50	-0.441%	
E	2637.021	47	-0.855%	
F	2793.826	45	+0.574%	
F#	2959.955	42	-0.548%	
G	3135.963	40	+0.350%	
G#	3322.438	38	+0.992%	
A	3520.000	36	+1.357%	
A#	3729.310	34	+1.417%	
B	3951.066	32	+1.134%	

NOTE	FREQUENCY	PERIOD	RELATIVE ERROR	Octave 4
C	4186.009	30	+0.462%	
C#	4434.922	28	-0.662%	
D	4698.636	27	+1.469%	
D#	4978.032	25	-0.441%	
E	5274.041	24	+1.246%	
F	5587.652	22	-1.685%	
F#	5919.911	21	-0.548%	
G	6271.927	20	+0.350%	
G#	6644.875	19	+0.992%	
A	7040.000	18	+1.357%	
A#	7458.621	17	+1.417%	
B	7902.133	16	+1.134%	

These values are all calculated from International A as follows :

```
FREQUENCY = 440*(2↑(OCTAVE+((N-10)/12)))
PERIOD = ROUND(125000/FREQUENCY)
```

where N is 1 for C, 2 for C#, 3 for D etc.

APPENDIX VIII - Cassette Reading and Writing.

1. BASIC Cassette Files.

BASIC reads and writes data in 2K byte blocks. Each block comprises one header record and up to eight data records. Each record has its own checksum. The header on each block contains the following information :

File Name
File Type
Block Number & first/last block indication
File length
Where written from (for Binary files)
Entry point (for Machine Code Binary files)

The file types recognised by BASIC are :

- BASIC program file
- BASIC protected program file
- ASCII text file
- Binary file

BASIC programs may be loaded from any of the first three file types, though there are some restrictions on BASIC protected files. The contents of Binary files may be read back into memory, and may be run.

The cassette file stream will only read and write ASCII text files.

2. Messages Produced by Cassette Handling Software.

Unless disabled the cassette handling software produces a number of messages while reading or writing, as follows :

```
Press PLAY then any key:
```

This message is issued when the software starts to read a tape (as a result of `LOAD`, `CAT`, `OPENIN` or cassette file reading).

The 'any key' is not quite accurate, pressing **[SHIFT]**, **[CTRL]** or **[CAPS LOCK]** has no effect. Pressing **[ESC]** will abandon the read operation.

`Found FILENAME block 9`

While searching for the required block of the required file to read some other file or block has been found. This message is for information only, but may indicate that the wrong tape is loaded, or that it is wound past the required block already.

`Rewind tape`

While searching for the required block of the required file to read a higher numbered block for that file has been found.

`Loading FILENAME block 9`

The required block of the required file has been found, and it is being loaded into memory.

`Press REC and PLAY then any key:`

This message is issued when the software starts to write a tape (as a result of **SAVE** or cassette file writing).

The 'any key' is not quite accurate, pressing **[SHIFT]**, **[CTRL]** or **[CAPS LOCK]** has no effect. Pressing **[ESC]** will abandon the write operation.

`Saving FILENAME block 9`

The given block of the given file is being written to tape.

`Read error x`

The 'x' is a single letter indicating what kind of read error has occured, as follows :

a = bit time out - rubbish data.
b = CRC error
d = block too long for buffer

`Write error a`

The software has failed while writing. Should never happen.

APPENDIX IV
Memory, Himem, Symbol After and Cassette Buffers.

1. Introduction to BASIC's Memory Usage

BASIC has a simple form of memory management. The user's program, arrays and variables are held at the bottom of the area from the lowest address available to BASIC up to the highest address available. The highest address available to BASIC is known as HIMEM, and is returned by the function of the same name.

The space immediately below HIMEM is occupied by the 'heap', in which strings are kept. As strings are added and processed the heap grows down toward the user's program, arrays etc. eating up the free space. During string processing some strings are discarded, but the memory released is not reclaimed until all the free space is exhausted and the heap is 'garbage collected'.

The lowest address available to BASIC is set when BASIC is first entered. HIMEM may be changed using the MEMORY command and is affected by SYMBOL AFTER commands and the use of the Cassette. When HIMEM is changed the heap is usually garbage collected, to ensure it is the minimum size possible, and then moved. If there are a lot of active strings the garbage collection may take an appreciable time.

2. MEMORY and HIMEM.

HIMEM gives the address of the highest byte in memory which BASIC may use for the user's BASIC program, variables, arrays and heap. The MEMORY command may be used to move HIMEM up and down, expanding or contracting the free space. BASIC will refuse a MEMORY command which it cannot obey, generating an error 7 (Memory full).

Moving HIMEM down reserves an area of memory, just above the new HIMEM, which BASIC will no longer attempt to use. This area may be used in any way the user wishes, but is typically used for assembler subroutines. The following creates an assembler subroutine to set the graphics paper:

```
10 MEMORY HIMEM-5                              :REM Allocate 5 bytes
20 gra.set.paper =HIMEM+1                      :REM Address of area allocated
30 POKE HIMEM+1,&DD:POKE HIMEM+2,&7E           :REM LD A,(IX+0)
40 POKE HIMEM+3,&C3:POKE HIMEM+4,&E4
                    :POKE HIMEM+5,&BB           :REM JP GRA SET PAPER
```

which may then be called thus:

```
50 CALL gra.set.paper,3:  REM Set graphics paper to ink 3.
```

Allocating areas of memory outside BASIC's control interacts with BASIC's own allocation of cassette buffers and user definable characters. as described below.

3. Cassette Buffers - Dynamic Allocation

While any cassette file is open BASIC requires 4K (4096) bytes of buffers. Rather than permanently reserve the space, making it unusable when no file is open, BASIC acquires it when required, by moving HIMEM down. When the buffer area is no longer required BASIC releases the space, if it can, by moving HIMEM up again. The buffer space can only be released if it is immediately above HIMEM. The following will effectively permanently allocate the buffers:

```
10 OPENOUT "dummy"      :REM Causes BASIC to allocate the buffers
20 as.you.was=HIMEM     :REM Remember current position
30 MEMORY HIMEM-1       :REM move HIMEM down one byte
40 CLOSEOUT             :REM Abandon the file
```

Note that nothing will be written to the tape, as BASIC does not bother to create empty files. The buffers cannot be released when the CLOSEOUT is obeyed because they are not immediately above HIMEM. The memory reserved by moving HIMEM since the buffers were allocated cannot be moved, so HIMEM cannot be moved, so the buffers are not released. If further cassette operations are performed the existing buffers are reused. The following would subsequently release the buffers:

```
50 MEMORY as.you.was
```

provided no cassette file were open. Returning the memory reserved by moving HIMEM removes the block on releasing the buffers.

4. SYMBOL AFTER - User Definable Characters.

The ROM contains matrices for all 256 characters available on the machine. The user may declare any number of characters as 'user definable', by stating the first character which is to be user definable. The SYMBOL AFTER command is the BASIC mechanism for declaring which characters are to be user definable, for example

```
SYMBOL AFTER 192
```

declares that all characters with numbers from 192 to 255 inclusive are to be user definable. The extreme cases are:

```
SYMBOL AFTER  0      :REM make all characters user definable
SYMBOL AFTER 256     :REM make no character user definable
```

The matrices for user definable characters must be moved from ROM into read/write memory so that they can be altered. BASIC allocates space for the matrices by changing HIMEM. If some user definable characters already exist a SYMBOL AFTER command may increase or decrease the memory required for matrices. BASIC can only do this if the current user definable matrices are immediately after HIMEM - it cannot reorganise the memory between HIMEM and the matrices.

When BASIC is first entered the last 16 characters (from 240 to 255) are implicitly declared as user definable. If more user definable characters are required and some memory is to be allocated above HIMEM, this may be achieved either by:

```
10 SYMBOL AFTER 160    :REM Make 92 user definable characters
20 MEMORY HIMEM-4096   :REM Allocate 4K bytes above HIMEM
```

or:

```
10 SYMBOL AFTER 256    :REM Dispose of all user definable characters
20 MEMORY HIMEM-4096   :REM Allocate 4K bytes above HIMEM
30 SYMBOL AFTER 192    :REM Make 64 user definable characters
```

The second scheme is more flexible, since it allows the program to change the number of user definable symbols later on.

INDEX TO CONTENTS

Keywords are indicated by UPPER case letters

ABS A.1
AFTER A.2
ASC A.3
ATN A.4
AUTO A.5
Address Expression 11.1
Arithmetic Functions 10.8
Array Variable Name 11.1
Arrays 2.4

BASIC Interrupts 9.1
BASIC Keywords (detail) A.1
BASIC Keywords (sumary) APP3.1
BIN$ B.1
BORDER B.2
Based Number 2.2
Binary Floating Point 2.9

CALL C.1
CAT C.2
CHAIN C.3
CHAIN MERGE C.3
CHR$ C.5
CINT C.6
CLEAR C.7
CLG C.8
CLOSEIN C.9
CLOSEOUT C.10
CLS C.11
CONT C.12
COS C.13
CREAL C.14
Call APP2.2
Cassette 10.5
Cassette Buffer APP9.2
Cassette Files 7.2
Cassette Handling Software APP8.1
Cassette Input 10.5
Cassette Reading APP8.1
Cassette Writing APP8.1
Character Set 2.1
Characters (user definable) .. 4.2
Characters (user defined) 10.8
Charaters (control) APP4.1
Colour 11.1
Colours 4.1
Colours APP6.1
Commands (external) 2.10
Commands 11.1
Control Structures 10.2
Coordinate System 4.3
Copy 6.1

DATA D.1
DEC$ D.2
DEF FN D.3
DEFINT D.4
DEFREAL D.4
DEFSTR D.4
DEG D.5
DELETE D.6
DI D.7
DIM D.8
DRAW D.9
DRAWR D.10
Data (constant) 10.6
Data 2.2
Data Format APP2.1
Decimal Fractions 2.9
Digit 1.2
Direct Mode 3.1
EDIT E.1
EI E.2
END E.3
ENT E.4
ENV E.6
EOF E.8
ERASE E.9
ERL E.10
ERR E.10
ERROR E.11
EVERY E.12
EXP E.13
Edit 6.1
Elements of Basic 2.1
Error Trapping 10.10
Errors APP1.1

Expression (logical) 2.8
Expression (numeric) 2.5
Expression (relational) 2.7
Expression (string) 2.7
Expression 2.5, 11.1
External Command APP2.2
External Routines and Commands APP2.1

FIX F.1
FOR F.2
FRE F.4
Frequency APP7.1
Function 2.8
Functions 11.1

GOSUB G.1
GOTO G.2
Graphics Screen 10.6
Graphics Screen 4.3

HEX$ H.1
HIMEM H.2
Himem APP9.1

IF I.1
INK I.2
INKEY I.3
INKEY$ I.4
INP I.5
INPUT I.6
INSTR I.8
INT I.9
Ink 11.1
Inks 4.1
Integer (unsigned) 2.5
Integer Expression 11.1
Interrupt Commands 9.2
Interrupts (timer) 10.7
Interrupts 9.1

JOY J.1
Joystick 10.7
Joystick 5.1
Joystick Key Numbers ... APP5.1

KEY DEF K.2
KEY K.1
Keyboard 10.7
Keyboard 5.1
Keyboard Commands 5.2
Keyboard Functions 5.2
Key numbers APP5.1
Keyboard and Joystick 5.1

LEFT$ L.1
LEN L.2
LET L.3
LINE INPUT L.4
LIST L.5
LOAD L.6
LOCATE L.7
LOG L.8
LOG10 ..... L.9
LOWER$ L.10
Line Input, Edit and Copy 6.1
Line Input 1.1
Line Input 6.1
Line Number 11.1
Loading 10.2
Logical Expression 11.2

MAX M.1
MEMORY M.2
MERGE M.3
MID$ M.4
MIN M.5
MODE M.6
MOVE M.7
MOVER M.8
Machine level operations .. 10.10
Masked Ink 11.2
Memory APP9.1
Memory APP9.1
Metalanguage 1.1
Mode 4.1

NEW N.1
NEXT N.2
Names 2.1
Notes APP7.1
Notes & Tone Periods ... APP7.1
Numbers 2.1
Numeric Expression 1.1
Numeric Expression 11.2
Numeric Variable 11.2

ON BREAK GOSUB O.2
ON BREAK STOP O.3
ON ERROR GOTO O.4
ON SQ GOSUB O.5

ON(exp)GOSUB O.1
ON(exp)GOTO O.1
OPENIN O.6
OPENOUT O.7
ORIGIN O.8
OUT O.9
Origin 4.3
Overflow APP1.1
Overview of Commands 10.1

PAPER P.1
PEEK P.2
PEN P.3
PI P.4
PLOT P.5
PLOTR P.6
POKE P.7
POS P.8
PRINT P.9
Place Holders 1.1
Program Creation 10.1
Program Development 10.10
Program Mode 3.1
Program Running 10.2
Program Termination 10.2

Quoted String 11.2

RAD R.1
RANDOMIZE R.2
READ R.3
RELEASE R.4
REM R.5
REMAIN R.6
RENUM R.7
RESTORE R.8
RESUME R.9
RETURN R.10
RIGHT$ R.11
RND R.12
ROUND R.13
RUN R.14
RUN R.15
Resolution 4.1
Rounding 2.4

SAVE S.1
SGN S.3
SIN S.4
SOUND S.5
SPACE$ S.7
SPEED INK S.8
SPEED KEY S.9
SPEED WRITE S.10
SQ S.11
SQR S.12
STOP S.13
STR$ S.14
STRING$ S.15
SYMBOL AFTER S.17
SYMBOL S.16
Scaled Number 2.2
Screen (text) 4.2
Screen 10.3
Screen 4.1
Screen Commands 4.4
Screen Functions 4.4
Simple Variable 11.3
Sound 8.1
Sound Commands 8.3
Sound Generation 10.6
Sound Generation 8.1
Stream (cassette) 10.5
Stream Expression 11.3
Streams (screen) 10.4
Streams 4.2
Streams 7.1
String Constant 2.2
String Expression 11.3
String Funtions 10.9
String Variable 11.3
Subroutine APP2.2
Symbol After APP9.2
SyntaxError APP1.1

TAG T.1
TAGOFF T.2
TAN T.3
TEST T.4
TESTR T.5
TIME T.6
TROFF T.7
TRON T.7
Text Input 10.5
Text Output 10.4
Text Screen Position APP4.1
The Screen 4.1
Timers 9.1

Tone Period APP7.1
Type Compatatbility 2.4
Type conversion functions ... 10.9
Type Marker 11.3

UNT U.1
UPPER$ U.2
Unscaled Number 2.1

VAL V.1
VPOS V.2
Variable 11.3
Variable Name 11.3
Variables 10.3
Variables 2.3

WAIT W.1
WEND W.2
WHILE W.3
WIDTH W.4
WINDOW SWAP W.6
WINDOW W.5
WRITE W.7
White Space 2.1
Window 4.3
Windows (text) 4.2

XPOS X.1
YPOS Y.1

ZONE Z.1